USING LEDs, LCDs AND GLCDs IN MICROCONTROLLER PROJECTS

www.it-ebooks.info

USING LEDs, LCDs AND GLCDs IN MICROCONTROLLER PROJECTS

Dogan Ibrahim
Near East University, Cyprus

A John Wiley & Sons, Ltd., Publication

www.it-ebooks.info

This edition first published 2012

© 2012, John Wiley & Sons, Ltd

Registered office

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

The right of the author to be identified as the author of this work has been asserted in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, except as permitted by the UK Copyright, Designs and Patents Act 1988, without the prior permission of the publisher.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book. This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data

Ibrahim, Dogan.
Using LEDs, LCDs, and GLCDs in microcontroller projects / Dogan Ibrahim.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-119-94070-8 (cloth)
1. Information display systems. 2. Liquid crystal devices–Automatic control. 3. Light emitting diodes–Automatic control. 4. Microcontrollers. I. Title.
TK7882.I6I185 2012
629.8'9–dc23

2012009481

A catalogue record for this book is available from the British Library.

Print ISBN: 9781119940708

Set in 10/12 pt TImes by Thomson Digital, Noida, India

www.it-ebooks.info

Contents

Preface **xiii**

Acknowledgements **xv**

1 Introduction to Microcontrollers and Display Systems **1**
1.1 Microcontrollers and Microprocessors 2
1.2 Evolution of the Microcontroller 3
1.3 Parts of a Microcontroller 4
1.3.1 Address 4
1.3.2 ALU 5
1.3.3 Analogue Comparator 5
1.3.4 Analogue-to-Digital Converter 5
1.3.5 Brown-out Detector 5
1.3.6 Bus 5
1.3.7 CAN 6
1.3.8 CISC 6
1.3.9 Clock 6
1.3.10 CPU 6
1.3.11 EEPROM 6
1.3.12 EPROM 6
1.3.13 Ethernet 7
1.3.14 Flash Memory 7
1.3.15 Harvard Architecture 7
1.3.16 Idle Mode 7
1.3.17 Interrupts 7
1.3.18 LCD Drivers 8
1.3.19 Pipelining 8
1.3.20 Power-on Reset 8
1.3.21 PROM 8
1.3.22 RAM 8
1.3.23 Real-time Clock 8
1.3.24 Register 9
1.3.25 Reset 9
1.3.26 RISC 9
1.3.27 ROM 9

www.it-ebooks.info

1.3.28 Serial Input-Output 9
1.3.29 Sleep Mode 9
1.3.30 Supply Voltage 10
1.3.31 Timers 10
1.3.32 USB 10
1.3.33 Watchdog 10
1.4 Display Devices 10
1.4.1 LED 10
1.4.2 7-Segment LED 11
1.4.3 OLED 12
1.4.4 LCD 12
1.5 Summary 15
Exercises 15

2 PIC18F Microcontrollers 17
2.1 The PIC18F2410 Microcontroller 18
2.2 PIC18F2410 Architecture 19
2.2.1 The Program Memory 21
2.2.2 The Data Memory 21
2.2.3 Power Supply Requirements 22
2.2.4 Oscillator Configurations 24
2.2.5 The Reset 30
2.2.6 Parallel I/O Ports 31
2.2.7 Timer Modules 38
2.2.8 Analogue-to-Digital Converter Module 43
2.2.9 Special Features of the CPU 48
2.2.10 Interrupts 49
2.2.11 Pulse Width Modulator Module 53
2.3 Summary 56
Exercises 56

3 C Programming Language 59
3.1 C Languages for Microcontrollers 59
3.2 Your First mikroC Pro for PIC Program 61
3.2.1 Comments 61
3.2.2 Beginning and Ending a Program 62
3.2.3 White Spaces 63
3.2.4 Variable Names 63
3.2.5 Reserved Names 64
3.2.6 Variable Types 64
3.2.7 Constants 66
3.2.8 Escape Sequences 68
3.2.9 Volatile Variables 69
3.2.10 Accessing Bits of a Variable 69
3.2.11 sbit Type 70
3.2.12 bit Type 70

www.it-ebooks.info

3.2.13 Arrays 70
3.2.14 Pointers 73
3.2.15 Structures 76
3.2.16 Unions 80
3.2.17 Operators in mikroC Pro for PIC 80
3.2.18 The Flow of Control 90
3.3 Functions in mikroC Pro for PIC 101
3.3.1 Function Prototypes 102
3.3.2 void Functions 103
3.3.3 Passing Parameters to Functions 104
3.3.4 Passing Arrays to Functions 106
3.3.5 Interrupt Processing 106
3.4 mikroC Pro for PIC Built-in Functions 108
3.5 mikroC Pro for PIC Libraries 109
3.5.1 ANSI C Library 109
3.5.2 Miscellaneous Library 111
3.6 Using the mikroC Pro for PIC Compiler 111
3.6.1 mikroC Pro for PIC IDE 112
3.6.2 Creating a New Source File 118
3.6.3 Compiling the Source File 122
3.7 Using the mikroC Pro for PIC Simulator 123
3.7.1 Setting a Break-Point 124
3.8 Other mikroC Pro for PIC Features 126
3.8.1 View Statistics 126
3.8.2 View Assembly 127
3.8.3 ASCII Chart 127
3.8.4 USART Terminal 127
3.8.5 Seven Segment Editor 127
3.8.6 Help 128
3.9 Summary 128
Exercises 129

4 PIC Microcontroller Development Tools – Including Display Development Tools 131
4.1 PIC Hardware Development Boards 132
4.1.1 Super Bundle Development Kit 132
4.1.2 PIC18 Explorer Board 132
4.1.3 PIC18F4XK20 Starter Kit 134
4.1.4 PICDEM 4 135
4.1.5 PIC16F887 Development Kit 135
4.1.6 FUTURLEC PIC18F4550 Development Board 137
4.1.7 EasyPIC6 Development Board 137
4.1.8 EasyPIC7 Development Board 139
4.2 PIC Microcontroller Display Development Tools 140
4.2.1 Display Hardware Tools 140
4.2.2 Display Software Tools 143

www.it-ebooks.info

4.3 Using the In-Circuit Debugger with the EasyPIC7 Development Board 145
4.4 Summary 149
Exercises 149

5 Light Emitting Diodes (LEDs) 151
5.1 A Typical LED 151
5.2 LED Colours 153
5.3 LED Sizes 154
5.4 Bi-Colour LEDs 154
5.5 Tri-Colour LEDs 155
5.6 Flashing LEDs 155
5.7 Other LED Shapes 155
5.8 7-Segment LEDs 156
5.8.1 Displaying Numbers 157
5.8.2 Multi-digit 7-Segment Displays 159
5.9 Alphanumeric LEDs 159
5.10 mikroC Pro for PIC 7-Segment LED Editor 163
5.11 Summary 163
Exercises 164

6 Liquid Crystal Displays (LCDs) and mikroC Pro for PIC LCD Functions 165
6.1 HD44780 Controller 165
6.2 Displaying User Defined Data 168
6.3 DDRAM Addresses 169
6.4 Display Timing and Control 171
6.4.1 Clear Display 172
6.4.2 Return Cursor to Home 172
6.4.3 Cursor Move Direction 172
6.4.4 Display ON/OFF 172
6.4.5 Cursor and Display Shift 173
6.4.6 Function Set 173
6.4.7 Set CGRAM Address 173
6.4.8 Set DDRAM Address 173
6.4.9 Read Busy Flag 174
6.4.10 Write Data to CGRAM or DDRAM 174
6.4.11 Read Data from CGRAM or DDRAM 174
6.5 LCD Initialisation 174
6.5.1 8-bit Mode Initialisation 175
6.5.2 4-bit Mode Initialisation 175
6.6 Example LCD Display Setup Program 177
6.7 mikroC Pro for PIC LCD Functions 180
6.7.1 Lcd_Init 180
6.7.2 Lcd_Out 181
6.7.3 Lcd_Out_Cp 181
6.7.4 Lcd_Chr 181

www.it-ebooks.info

6.7.5 Lcd_Chr_Cp 181
6.7.6 Lcd_Cmd 182
6.8 Summary 182
Exercises 183

7 Graphics LCD Displays (GLCD) **185**
7.1 The 128 × 64 Pixel GLCD 185
7.2 Operation of the GLCD Display 187
7.3 mikroC Pro for PIC GLCD Library Functions 189
7.3.1 Glcd_Init 189
7.3.2 Glcd_Set_Side 190
7.3.3 Glcd_Set_X 190
7.3.4 Glcd_Set_Page 190
7.3.5 Glcd_Write_Data 190
7.3.6 Glcd_Fill 190
7.3.7 Glcd_Dot 191
7.3.8 Glcd_Line 191
7.3.9 Glcd_V_Line 191
7.3.10 Glcd_H_Line 191
7.3.11 Glcd_Rectangle 192
7.3.12 Glcd_Rectangle_Round_Edges 192
7.3.13 Glcd_Rectangle_Round_Edges_Fill 192
7.3.14 Glcd_Box 193
7.3.15 Glcd_Circle 193
7.3.16 Glcd_Circle_Fill 194
7.3.17 Glcd_Set_Font 194
7.3.18 Glcd_Set_Font_Adv 194
7.3.19 Glcd_Write_Char 195
7.3.20 Glcd_Write_Char_Adv 195
7.3.21 Glcd_Write_Text 195
7.3.22 Glcd_Write_Text_Adv 195
7.3.23 Glcd_Write_Const_Text_Adv 196
7.3.24 Glcd_Image 196
7.4 Example GLCD Display 196
7.5 mikroC Pro for PIC Bitmap Editor 198
7.6 Adding Touch-screen to GLCDs 199
7.6.1 Types of Touch-screen Displays 200
7.6.2 Resistive Touch Screens 200
7.7 Summary 203
Exercises 204

8 Microcontroller Program Development **205**
8.1 Using the Program Description Language and Flowcharts 205
8.1.1 BEGIN – END 206
8.1.2 Sequencing 206

www.it-ebooks.info

8.1.3 IF – THEN – ELSE – ENDIF 206
8.1.4 DO – ENDDO 207
8.1.5 REPEAT – UNTIL 209
8.1.6 Calling Subprograms 209
8.1.7 Subprogram Structure 209
8.2 Examples 211
8.3 Representing for Loops in Flowcharts 216
8.4 Summary 218
Exercises 218

9 LED Based Projects 219
9.1 PROJECT 9.1 – Flashing LED 219
9.2 PROJECT 9.2 – Binary Counting Up LEDs 226
9.3 PROJECT 9.3 – Rotating LEDs 229
9.4 PROJECT 9.4 – Wheel of Lucky Day 231
9.5 PROJECT 9.5 – Random Flashing LEDs 239
9.6 PROJECT 9.6 – LED Dice 240
9.7 PROJECT 9.7 – Connecting more than one LED to a Port Pin 246
9.8 PROJECT 9.8 – Changing the Brightness of LEDs 250
9.9 PROJECT 9.9 – LED Candle 264
9.10 Summary 267
Exercises 267

10 7-Segment LED Display Based Projects 269
10.1 PROJECT 10.1 – Single Digit Up Counting 7-Segment LED Display 269
10.2 PROJECT 10.2 – Display a Number on 2-Digit 7-Segment LED Display 271
10.3 PROJECT 10.3 – Display Lottery Numbers on 2-Digit 7-Segment LED Display 278
10.4 PROJECT 10.4 – Event Counter Using 4-Digit 7-Segment LED Display 285
10.5 PROJECT 10.5 – External Interrupt Based Event Counter Using 4-Digit 7-Segment LED Display with Serial Driver 292
10.6 Summary 302
Exercises 303

11 Text Based LCD Projects 305
11.1 PROJECT 11.1 – Displaying Text on LCD 305
11.2 PROJECT 11.2 – Moving Text on LCD 307
11.3 PROJECT 11.3 – Counting with the LCD 310
11.4 PROJECT 11.4 – Creating Custom Fonts on the LCD 315
11.5 PROJECT 11.5 – LCD Dice 317
11.6 PROJECT 11.6 – Digital Voltmeter 325
11.7 PROJECT 11.7 – Temperature and Pressure Display 327
11.8 PROJECT 11.8 – The High/Low Game 333
11.9 Summary 344
Exercises 345

www.it-ebooks.info

12 Graphics LCD Projects **347**
12.1 PROJECT 12.1 – Creating and Displaying a Bitmap Image 347
12.2 PROJECT 12.2 – Moving Ball Animation 355
12.3 PROJECT 12.3 – GLCD Dice 357
12.4 PROJECT 12.4 – GLCD X-Y Plotting 372
12.5 PROJECT 12.5 – Plotting Temperature Variation on the GLCD 374
12.6 PROJECT 12.6 – Temperature and Relative Humidity Measurement 385
12.7 Operation of the SHT11 386
12.8 Acknowledgement 389
12.9 Summary 400
Exercises 400

13 Touch Screen Graphics LCD Projects **401**
13.1 PROJECT 13.1 – Touch Screen LED ON-OFF 401
13.2 PROJECT 13.2 – LED Flashing with Variable Rate 410
13.3 Summary 418
Exercises 418

14 Using the Visual GLCD Software in GLCD Projects **419**
14.1 PROJECT 14.1 – Toggle LED 420
14.2 PROJECT 14.2 – Toggle more than One LED 425
14.3 PROJECT 14.3 – Mini Electronic Organ 426
14.4 PROJECT 14.4 – Using the SmartGLCD 430
14.5 PROJECT 14.5 – Decimal to Hexadecimal Converter using the SmartGLCD 444
14.6 Summary 452
Exercises 452

15 Using the Visual TFT Software in Graphics Projects **453**
15.1 PROJECT 15.1 – Countdown Timer 454
15.2 PROJECT 15.2 – Electronic Book 462
15.3 PROJECT 15.3 – Picture Show 467
15.4 Summary 472
Exercises 472

Bibliography **473**

Index **475**

www.it-ebooks.info

Preface

A microcontroller is a single chip microprocessor system, which contains data and program memory, serial and parallel I/O, timers, and external and internal interrupts, all integrated into a single chip that can be purchased for as little as £2.00. About 40% of microcontroller applications are in office automation, such as PCs, laser printers, fax machines, intelligent telephones, and so on. About one-third of microcontrollers are found in consumer electronic goods. Products such as CD players, hi-fi equipment, video games, washing machines, cookers and so on fall into this category. The communications market, automotive market and the military share the rest of the application areas.

Input and output are very important parts of any microcontroller system. Typical input devices are push-button switches, keypads and various analog and digital sensors. Typical output devices are Light Emitting Diodes (LEDs), Liquid Crystal Displays (LCDs), Graphics Liquid Crystal Displays (GLCDs), motors, actuators, buzzers, and so on. This book is about the theory and applications of display devices in microcontroller based systems. The book explains briefly the theory of the commonly used display devices, namely LEDs, 7-Segment LED displays, LCDs, monochrome GLCDs and TFT based colour LCDs. In addition, the use of each display device is explained with several working and tested projects. The description, block diagram, circuit diagram, operation and full program code of all the projects are given. PIC18F series of high-end microcontrollers are used in all the projects. The projects are developed using the highly popular mikroC Pro for PIC compiler. Knowledge of the C programming language will be useful. Also, familiarity with at least one member of the PIC16F series of microcontrollers will be an advantage. The knowledge of assembly language programming is not required because all the projects in the book are based on using the C language.

This book is written for students, for practising engineers and for hobbyists interested in developing display based projects using the PIC series of microcontrollers.

Chapter 1 presents the basic features of microcontrollers and the basic features of display devices used in such systems.

Chapter 2 provides a review of the PIC18 series of microcontrollers. Various features of these microcontrollers are described in detail. The PIC18F2410 is chosen as a typical microcontroller.

Chapter 3 provides a short tutorial on the C language and then examines the features of the mikroC Pro for PIC compiler used in PIC series of microcontrollers.

Chapter 4 is about the important topic of microcontroller development tools. Both the software and hardware development tools are described in detail. In addition, the use of microcontroller simulators and in-circuit debuggers are described with examples.

www.it-ebooks.info

Chapter 5 provides the basic theory of LEDs. The use of simple LEDs and 7-Segment simple and multiplexed LEDs are explained with examples.

Chapter 6 provides some simple projects using the PIC18 series of microcontrollers and the mikroC Pro for PIC C language compiler. All the projects in this chapter are based on the PIC18F452 microcontroller and all the projects have been tested and are working. This chapter should be useful for those who are new to PIC microcontrollers, and for those who want to extend their knowledge of programming PIC18F series of microcontrollers using the mikroC Pro for PIC language.

Chapter 7 covers the theory of LCD displays. The basic working principles of LCDs and the mikroC Pro for PIC built-in LCD functions are explained with several examples.

Chapter 8 is about the Program Development Language (PDL) used to describe the operation of software in general. Various building blocks of the PDL are described in this chapter.

Chapter 9 provides simple LED based projects, ranging from LED flashing to more complex LED projects.

Chapter 10 is about 7-Segment LED based projects. Several single digit and multiplexed working and tested projects are given in this chapter with full source code.

Chapter 11 provides several text based LCD projects. The use of LCDs is described in this chapter through simple and complex projects, ranging from displaying simple text on an LCD to developing an LCD based voltmeter project.

Chapter 12 is about the use of GLCDs in microcontroller projects. The use of standard monochromatic 128×64 pixel GLCD is used in the projects in this chapter.

Touch screen displays are important application areas of microcontrollers. Chapter 13 gives several projects on using touch screens in graphics applications.

The Visual GLCD software package is used for the development of projects based on several different types of monochromatic GLCD displays. Chapter 14 explains the use of this software package and gives the steps required to develop GLCD based applications. Several projects are given in this chapter using the Visual GLCD software package with both 128×64 pixel and 240×128 pixel GLCD displays.

Finally, Chapter 15 is about the Visual TFT software package used for the development of TFT based colour graphics applications. The chapter describes the steps required to create microcontroller based TFT graphics applications using the MikroMMB graphics development board.

Dogan Ibrahim
London, 2012

www.it-ebooks.info

Acknowledgements

The following material is reproduced in this book with the kind permission of the respective copyright holders and may not be reprinted, or reproduced in any way, without their prior consent.

Figures 2.1–2.6, 2.10, 2.11, 2.13, 2.17, 2.28, 2.30, 2.32–2.37 are taken from Microchip Technology Inc. Data Sheet PIC18F2X1X/4X1X (DS39636D). Figures 4.2–4.4 are taken from the web site of Microchip Technology Inc.

Figure 4.1 is taken from the web site of microEngineering Labs Inc.

Figure 4.5 and 4.6 are taken from the web site of Custom Computer Services Inc.

Figure 4.7 is taken from the web site of Futurlec Inc.

Figures 4.8 and 4.9 are taken from the web site of mikroElektronica.

PIC®, PICSTART® and MPLAB® are all trademarks of Microchip Technology Inc.

www.it-ebooks.info

1

Introduction to Microcontrollers and Display Systems

The basic building blocks of any digital computer are the central processing unit (CPU), the memory and the input-output (I/O). The CPU is like the human brain, as it controls all internal operations of the computer. Instructions are fetched from the memory under the control of the CPU, which it then decodes and controls various internal parts of the computer so that the required operations are performed. The CPU also includes an arithmetic and logic unit (ALU), which is used to perform mathematical and logical operations. The result of an operation is stored either in the memory, in a temporary register, or is sent to an I/O port. Two types of memories are used in a computer, as far as memory functionality is concerned. The program memory stores the user instructions and this memory is normally non-volatile, that is the data is not lost after removal of the power. The second type of memory is the data memory, which stores the temporary user data, such as the result of an operation. The I/O ports allow the computer to communicate with the external world. For example, a keyboard is an input device, enabling the user to enter data to the computer. Similarly, a printer is an output device, enabling the user to print out a hard copy of data in paper form. Depending on the actual application and the requirements, a computer may include additional components, such as timers, counters, interrupt logic, clock logic, and so on.

A computer program consists of a collection of instructions for performing a specific task. In the early days of computers, programs were written in Assembly language, which was a short way of specifying instructions using words called mnemonics. Although Assembly language was fast, it had several disadvantages. Writing a long and complex program using Assembly language was difficult. More importantly, it was difficult to maintain a program written in Assembly language. Also, different processors had different instruction sets and different Assembly language instructions, resulting in no portability. Consequently, it was a tedious task to convert a program written for one processor to function on another processor. Over the last decade, nearly all programs have been written using a high level language such as C, BASIC or Pascal. High level languages have several advantages. First, learning to program in a high level language is easy. Second, the developed code is highly portable. For example, a C program written for a processor can easily be modified to work on another type

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

of processor. This is true, even if the two processors are manufactured by different vendors. Third, high level programs are much easier to develop and maintain.

1.1 Microcontrollers and Microprocessors

A microcontroller is basically a single chip computer, generally requiring no external components. A microprocessor differs from a microcontroller in many ways. Perhaps the main difference is that a microcontroller can function as a computer without the need of any external hardware. A microprocessor, on the other hand, is just the CPU of a computer, and requires several other external components before it becomes a useful computer. Because a microcontroller consists of a single chip, its power consumption is low. The development of a microcontroller based system is also easy, as the processing hardware consists of a single chip. Perhaps the only advantage of a microprocessor over a microcontroller is that a microprocessor can easily be expanded to have more memory or I/O. The expansion of microcontrollers is more difficult and a different model is usually chosen when higher performance, more memory or more I/O are required.

Figure 1.1 shows the structure of a computer, built using a microprocessor. Here the hardware consists of several components, all attached to the microprocessor chip. The structure of a microcontroller based computer is shown in Figure 1.2. The advantages of using a microcontroller instead of a microprocessor are clear when Figures 1.1 and 1.2 are compared.

The differences between a microprocessor and a microcontroller are summarised below:

- A microprocessor is a single chip CPU microcontroller containing a CPU, memory, I/O, timers, counters and much of the remaining circuitry of a complete computer system on a single chip.
- The power consumption of a microprocessor based computer is very large, in the order of amperes. On the other hand, the power consumption of a microcontroller based computer is in the range of several hundred milliamperes. In addition, microcontrollers can be operated in sleep modes, which consume currents as low as tens of nanoamperes.
- A microprocessor based computer costs much more than a microcontroller based system.

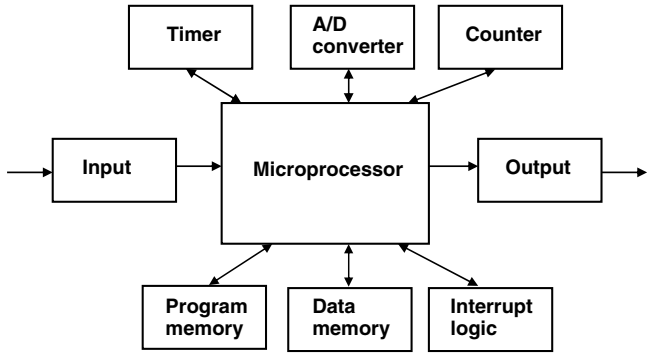


Figure 1.1 Structure of a microprocessor based computer

www.it-ebooks.info

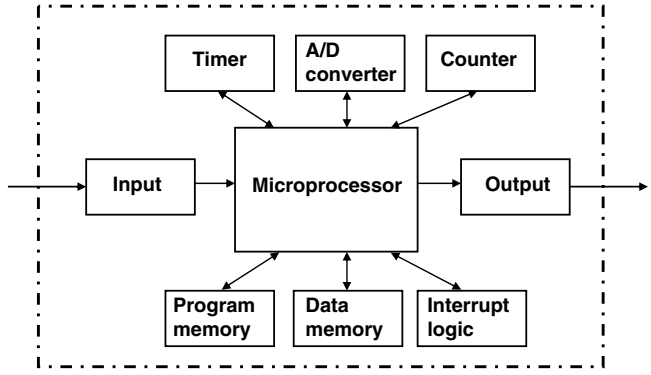


Figure 1.2 Structure of a microcontroller based computer

- Because a microcontroller based system consists of a single chip, it has higher reliability.
- Microprocessor based systems can easily be expanded, for example by adding more memory or I/O chips. It is usually not possible to expand a microcontroller system. If an application requires more memory, more I/O or higher processing power, then a different model microcontroller is usually chosen.

Although microcontrollers have only been with us for a few decades, they have been used in many consumer, commercial, industrial and educational devices. Some examples are found in:

- *Offices*: in typewriters, computers, calculators, photocopiers, scanners, plotters, elevators, and so on;
- *Homes*: in microwave ovens, washing machines, alarm clocks, dish washers, hi-fi equipment, DVD players, digital televisions, and so on;
- *Industry*: in automatic control systems, safety systems, robotics, motor control, and so on;
- *Transportation systems*: in vehicles, traffic signals, road signs, speed cameras, GPS systems, and so on;
- *Supermarkets*: in weighing scales, cash registers, electronic signs, card readers, and so on;
- *Play*: in electronic toys, MP3 players, video games, mobile phones, and so on;
- *Education*: in electronic white-boards, photocopiers, projectors, calculators, and so on.

1.2 Evolution of the Microcontroller

The first microprocessor, named the 4004, was introduced by the Intel Corporation in 1971. This was a simple 4-bit device, supported by three other chips to make a computer; the 4001 and 4002 memory chips, and the 4003 shift register. 4004 was initially used in calculators and in simple control applications.

Shortly after the 4004 appeared in the commercial marketplace, many electronic companies realised the power and future prospects of microprocessors and so have heavily invested in this field. Three other general-purpose microprocessors were soon introduced: Rockwell International 4-bit PPS-4, Intel 8-bit 8008 and the National Semiconductor 16-bit IMP-16.

www.it-ebooks.info

These microprocessors were based on PMOS technology and can be classified as the first-generation devices.

In the early 1970s, we see the second-generation microprocessors in the marketplace, designed using the NMOS technology. The shift to NMOS technology resulted in higher execution speeds, as well as higher chip densities. During this time, we see 8-bit microprocessors such as the Motorola 6800, Intel 8080 and 8085, the highly popular Zilog Z80, and Motorola 6800 and 6809.

The third generation of microprocessors were based on HMOS technology, which resulted in higher speeds and, more importantly, higher chip densities. During 1978, we see the 16-bit microprocessors such as the Intel 8086, Motorola 68 000 and Zilog Z8000. The 8086 microprocessor was so successful that it was used in early PC designs (called PC XT).

The fourth generation of microprocessors appeared around the 1980s and the technology was based on HCMOS. During this generation we see the introduction of 32-bit devices into the marketplace. Intel introduced the highly popular 32-bit microprocessors 80 386, 80 486, and the Pentium family; and Motorola introduced the 68 020 family. The Intel processors have been used heavily in early PC designs. In parallel to the development of 32-bit microprocessors, we see the introduction of early single chip computers (later named microcontrollers) into the marketplace. The Intel 8048 was the first microcontroller, followed by the highly popular 8051 series. The 8051 device has been so popular that it is still in use today. This device was a true single chip computer, containing a CPU, data memory and erasable program memories, I/O module, timer/counter, interrupt logic, clock logic, and serial communications module, such as the Universal Synchronous Asynchronous Receiver Transmitter (USART). After the success of the 8051, we see many other companies offering microcontrollers. Today, some of the most popular general-purpose low-cost 8-bit microcontrollers are Microchip PIC series, Atmel AVR series, Motorola HC11 series, and 8051 and its derivatives.

The fifth and the current generation of microcontrollers are now based on 16-bit and 32-bit architectures (e.g. PIC32 series). It is interesting to note that currently the 8-bit microcontrollers are still popular and much more in demand. This is because of their simple architectures, low cost, low power requirements, and the availability of the vast number of hardware and software development tools. The power offered by the high-end 8-bit microcontrollers (e.g. the PIC18F series) are enough for most medium to high-speed applications, except perhaps in special cases of digital signal processing where much higher throughput is generally required.

1.3 Parts of a Microcontroller

Before explaining microcontroller architectures and programming, it is worthwhile to look at the parts of a microcontroller in more detail and understand some basic terms.

1.3.1 Address

Address is a binary pattern that is used to represent memory locations. An address bus is a collection of address lines in a processor. For example, most 8-bit microcontrollers have a 16-bit address bus, capable of addressing up to 65 536 different memory locations (0 to 65 535).

www.it-ebooks.info

1.3.2 ALU

An arithmetic and logic unit (ALU) is part of a computer where the actual mathematical and logical operations are performed. 8-bit microcontrollers have 8-bit ALU modules. Typical operations carried out by an ALU are addition, subtraction, division, logical ANDing, ORing, Exclusive-OR and comparison. Some ALUs can also perform signed or unsigned multiplication.

1.3.3 Analogue Comparator

Some microcontrollers have built-in analogue comparator modules. An analogue comparator module is used to compare the voltage levels of two analogue signals. Although this feature is implemented in most mid-range PIC microcontrollers, it is not an important functionality.

1.3.4 Analogue-to-Digital Converter

Analogue-to-digital converter (A/D converter) is used to convert an analogue input signal into digital form, so that the signal can be processed within the microcontroller. Most mid-range PIC microcontrollers have built-in A/D converter modules. In general purpose and low-speed applications, the A/D converters are 8 to 10 bits, having 256 or 1024 quantisation levels. An A/D converter can either be unipolar or bipolar. Unipolar converters can only handle signals that are always positive. Bipolar converters, on the other hand, can handle both positive and negative signals. The A/D converters implemented in PIC series of microcontrollers are unipolar. The A/D conversion process is started by the user program and the conversion can take tens of processor cycles to complete. The user program has the option of either polling the conversion status and waiting until the conversion is complete, or alternatively, the A/D converter completion interrupt can be enabled to generate an interrupt as soon as the conversion is complete.

1.3.5 Brown-out Detector

Brown-out detectors in microcontrollers is a feature that can be configured to reset a microcontroller if the power supply voltage falls below a nominal value. The brown-out detector is a safety feature, as it protects the microcontroller data or the program from being corrupted while working below the recommended supply voltage.

1.3.6 Bus

A bus is a collection of wires grouped together in terms of their functions. An 8-bit conventional microprocessor usually has three buses: address bus, data bus and control bus. Memory and I/O addresses are sent over the uni-directional address bus. Data and instructions from the memory are sent over the bi-directional data bus. Processor control signals are sent over the uni-directional control bus. Some microprocessors have an additional I/O bus, where the I/O device addresses are sent.

www.it-ebooks.info

1.3.7 CAN

CAN bus is used in the automotive industry. Some microcontrollers include CAN bus modules, which simplify the design of CAN bus based products. For example, the PIC18F4680 provides CAN interface.

1.3.8 CISC

CISC is also known as the Complex Instruction Computer. In CISC architecture, both data and instructions are of the same width (e.g. 8-bits wide) and the microcontroller usually has over 200 instructions. Data and instructions are on the same bus and cannot be fetched at the same time.

1.3.9 Clock

A clock is basically a square wave signal used to provide timing signals to a digital processor. A clock is generated either using external devices (e.g. crystal, resistor-capacitor etc.), or some microcontrollers have built-in clock generation circuits. The PIC18F microcontroller family can operate with clock frequencies of up to 40 MHz. The basic instruction cycle in a PIC microcontroller takes four clock cycles. Thus, the effective operating frequency, or the MIPS (Millions of Instructions per Second) value is equal to the clock frequency divided by four, that is 10 MIPS.

1.3.10 CPU

The central processing unit (CPU), is the brain of a computer system, administering all activity in the system and performing all operations on data. The CPU consists of the ALU, several registers, and the control and synchronisation logic. The CPU fetches instructions from memory, decodes these instructions, and finally executes them. Decoding an instruction is the process of deciding what control signals to send to other internal parts of the computer for the successful execution of the instruction.

1.3.11 EEPROM

The electrically erasable programmable read only memory (EEPROM) is a non-volatile memory that can be erased and reprogrammed using a suitable programming device. EEPROMs are used in microcontroller based systems to store semi-permanent data, such as configuration data, maximum and minimum values, identification data, setup data, and so on. Most PIC microcontrollers have built-in EEPROM memories. One disadvantage of these memories is their much slower write times than their read times.

1.3.12 EPROM

The erasable programmable read only memory (EPROM) can be programmed and erased. An EPROM memory chip has a small clear-glass window on top of the chip, where the data

www.it-ebooks.info

can be erased under strong ultraviolet light in a few minutes. An EPROM is programmed by inserting the chip into a socket of an EPROM programmer device, which is connected to a PC. After programming the chip, the window can be covered with dark tape to prevent accidental erasure of the data, for example under direct sunlight. An EPROM must be erased before it can be re-programmed. EPROM memories are commonly used during the program development time where the programs keep changing until finalised. Some versions of EPROMs are known as One Time Programmable (OTP), which can be programmed only once but cannot be erased.

1.3.13 Ethernet

The Ethernet interface enables a microcontroller to be connected to a local area network, and in addition provides Ethernet interface capabilities. A microcontroller with such an interface can be connected to the Internet and can send and receive TCP/IP based packets. Some microcontrollers, such as the PIC18F97J60, have built-in Ethernet capabilities.

1.3.14 Flash Memory

Flash memory is a non-volatile memory used mainly to store user programs. This type of memory can be programmed electrically while embedded on the board. Some microcontrollers have only 1 KB flash memory, while some others can have 32 KB or more. In addition to computers, flash memory is also used in mobile phones and digital cameras.

1.3.15 Harvard Architecture

This is a type of CPU where the program memory and data memory units and buses are separate. The result is that the processor can fetch instructions and data at the same time, thus increasing the performance. Several microcontrollers, including the PIC family, are designed using the Harvard architecture.

1.3.16 Idle Mode

This mode is similar to the sleep mode and is used to conserve power. In idle mode, the internal oscillator is off but the peripheral devices are on.

1.3.17 Interrupts

Interrupts cause a microcontroller to respond to external or internal events in the shortest possible time. An internal interrupt usually comes from the timer module, where an interrupt can be generated whenever a timer overflows. Thus, events can be scheduled to happen at regular intervals. External interrupts usually come from the microcontroller I/O ports. For example, the microcontroller can be configured to create an interrupt when the state of a port pin changes its value. When an interrupt occurs, the microcontroller leaves its normal flow of program execution and jumps to the Interrupt Service Routine (ISR). At this point the code inside the ISR is executed and at the end of this code the program returns and continues to

www.it-ebooks.info

execute the code just before the interrupt occurred. The ISR is usually at a fixed address of the program memory, known as the interrupt vector address. Some microcontrollers have priority based interrupt sources, with different interrupt vector addresses for different sources.

1.3.18 LCD Drivers

Some microcontrollers offer LCD drivers and interface signals, so that standard LCD modules can be directly connected. Since all of the LCD functions can be implemented in software, such microcontrollers are not popular.

1.3.19 Pipelining

Pipelining is a technique used in computer systems to overlap the instruction fetch time with execution time. This allows higher throughput as two operations are performed in parallel. In microcontrollers, pipelining is generally used to fetch the next instruction while executing the current instruction. PIC microcontrollers use two-stage pipelining to speed up the execution time.

1.3.20 Power-on Reset

The power-on reset circuit keeps the microcontroller in the reset state until all the internal circuitry has been initialised. This is important, as it places the microcontroller clock into a known state. The power-on reset can be enabled or disabled during programming of PIC microcontrollers.

1.3.21 PROM

Programmable read only memory (PROM) is a non-volatile memory similar to a ROM. But PROM can be programmed by the end user with the aid of a PROM programmer device. PROM can only be programmed once and its contents cannot be changed after programming the device.

1.3.22 RAM

Random access memory (RAM) is a general purpose read-write memory used to store temporary data in a program. RAM is a volatile memory where the stored data is cleared after the power is turned off. All microcontrollers have some amount of RAM. Some may have only a few hundred bytes, while others can have up to 4 KB or more.

1.3.23 Real-time Clock

A real-time clock enables a microcontroller to receive absolute date and time information. Some microcontrollers have built-in hardware real-time clock modules. In general, an external real-time clock chip can be connected to general purpose microcontroller I/O ports to receive the absolute date and time information.

www.it-ebooks.info

1.3.24 Register

A register is a volatile, temporary high-speed storage for data. All microcontrollers have some amount of registers. Some microcontrollers, such as the PIC family, have a Special Function Register (SFR), used to hold the configuration data for various functions of the microcontroller. For example, the I/O direction registers hold the direction of each I/O pin. Similarly, the PORT registers hold the data received from a port, or data to be sent to an I/O port.

1.3.25 Reset

All microcontrollers have reset facilities. A reset action can be automatic by software (e.g. when the watchdog is enabled but not refreshed), or an external button can be used to reset the microcontroller. Reset puts the microcontroller into a known state. Usually, after a reset, the program starting from memory address 0 of the microcontroller is executed.

1.3.26 RISC

In a Reduced Instruction Set Computer (RISC) microcontroller, the data and instructions are not usually of the same width. For example, in an 8-bit RISC microcontroller, the data is 8-bits but the instructions can be 12, 14 or 16 bits wide. RISC microcontrollers have a limited number of instructions (e.g. not more than 50).

1.3.27 ROM

Read only memory (ROM) is non-volatile and is used to store user programs. A ROM is normally programmed in the factory during the manufacturing process. ROM is not reprogrammable and its contents cannot be erased. ROM is normally used when a program has been tested and is working correctly, and it is desired to make thousands of copies of the same program.

1.3.28 Serial Input-Output

Serial ports on a microcontroller enable communication using the RS232 protocol. For example, the microcontroller can be connected to a PC via its serial port and then data can be exchanged between the microcontroller and the PC. Although serial communication can be implemented in software, most microcontrollers have built-in USART modules to read and write serial data through its ports. Most mid-range PIC microcontrollers are equipped with at least one USART module.

1.3.29 Sleep Mode

Some microcontrollers have built-in sleep modes where, in this mode, the internal oscillator is stopped. The reason for using this mode is to reduce the power consumption to a very low level. In this mode all the microcontroller internal circuitry and the peripheral devices are in the off state. The microcontroller is usually woken up from sleep mode by an external reset or a watchdog time-out.

www.it-ebooks.info

1.3.30 Supply Voltage

Most microcontrollers operate with the standard logic voltage of +5 V. The range of acceptable voltage is usually in the range +4.75 to +5.25 V. The manufacturers' data sheets usually give the acceptable power supply voltage limits. PIC18F microcontrollers can operate with a power supply of +2 to +5.5 V. The required power supply voltage is usually obtained using a regulated power supply. In portable applications, the +5 V supply is obtained using a +9 V battery with a +5 V regulator chip (e.g. 78L05).

1.3.31 Timers

Timers are used in timing and counting applications. Most microcontrollers are equipped with at least one, and in many cases, several timers. A timer is usually 8 or 16 bits wide. Data is loaded into the timer under program control. The timer counts up at each clock pulse (or every time an external event occurs), and when the timer overflows an interrupt is generated (if interrupts are enabled). One common application of timers is to generate delays in programs, or to schedule events at regular intervals.

1.3.32 USB

USB is a powerful high-speed communications port used to connect various devices together. Some microcontrollers include built-in USB modules, which simplify the USB based communications. For example, the PIC18F2 × 50 microcontroller has a built-in USB module.

1.3.33 Watchdog

A watchdog is basically a programmable timer circuit that can be refreshed by the user program. It is usually used in real-time, and time based applications where time critical modules of a program are used to refresh the watchdog. If the watchdog fails to be refreshed, this is a sign that a time critical module has not completed its task. An automatic software reset occurs if the watchdog is enabled but is not refreshed. The watchdog is a safety feature, used to detect loops and runaway code in programs.

1.4 Display Devices

Display devices are output devices that can be connected to I/O ports of microcontrollers. Most electronic equipment, whether consumer related, commercial or industrial, have some form of display device, for example, mobile phones, calculators, GPS systems, printers, computers, MP3 players, microwave ovens, and so on.

In this section we are only concerned with small display devices commonly used in microcontroller based projects. In general, we can divide these display devices into three groups: LED based, OLED based and LCD based.

1.4.1 LED

Light Emitting Diode (LED) based displays are further divided into two groups: Simple LED based and 7-segment LED based. Simple LED devices (see Figure 1.3) consist of a single or an array of LEDs, commonly used in applications to indicate the status of something, for

www.it-ebooks.info

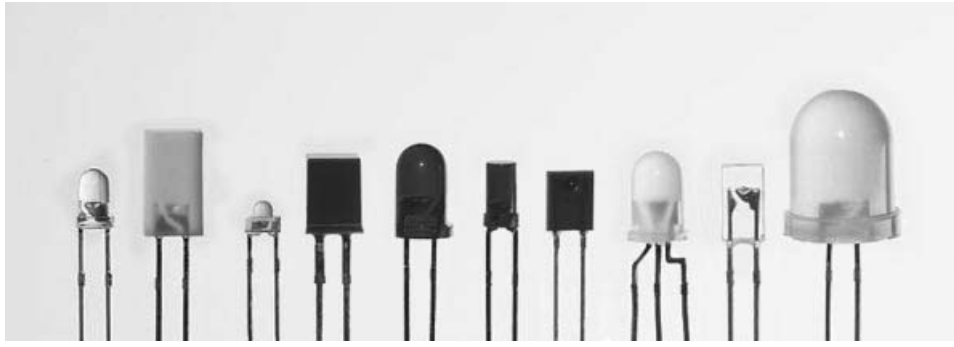


Figure 1.3 Simple LEDs

example, the on/off status of an electronic device, the selection of an item, and so on. Simple LEDs are available in various colours, such as red, green, orange, blue and white, and are directly connected to I/O ports of microcontrollers via current limiting resistors.

1.4.2 7-Segment LED

7-segment LEDs (see Figure 1.4) are generally used to display numeric data. The numbers are made up of 7 segments and the required number is displayed by turning on or off the appropriate segments. There are two types of 7-segment displays: common-anode or common-cathode. In common-anode displays, the anode pin is connected to the supply voltage and the individual segments are turned on by grounding the required segment. In a common-cathode type display, the cathode is connected to ground and the individual segments are turned on by applying voltage to the required segment. Both types can easily be connected and driven from a microcontroller I/O pin. To display numbers between 0 and 9, a single digit is used. To display higher numbers, it is necessary to use multiple digits (see Figure 1.5). In multi-digit applications, each digit is turned on or off by controlling its

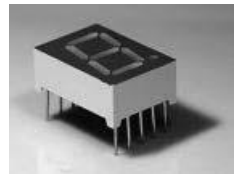


Figure 1.4 7-segment display

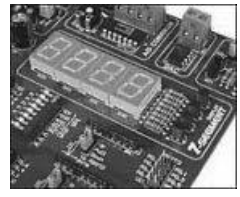


Figure 1.5 7-segment multiplexed 4-digit display

www.it-ebooks.info

common pin. The digits are enabled and disabled alternately, and very fast in such a way that when viewed the user thinks that the display is stationary.

1.4.3 OLED

Organic Light Emitting Diode (OLED) displays can be used to display text as well as graphical images. These displays are constructed by inserting organic material between a pair of electrodes where at least one of the electrodes is transparent. When an electric current is applied to the two conductors, a bright, electro-luminescent light is produced from the organic material. There are two types of OLEDs, as far as the used material is concerned: those based on small molecules and those employing polymers. OLED displays work without a backlight and thus they can be used both outdoors and indoors in low ambient light conditions.

OLEDs have several advantages compared to other displays:

- OLEDs have wide viewing angles and improved brightness. The pixel colours appear correct, even as the viewing angle approaches vertical from normal.
- OLED displays have very fast response times, more than 200 times faster than LCDs.
- OLED displays can be fabricated on flexible substrates, with the possibility of making roll-up displays embedded in fabrics.
- OLED displays produce sharp and bright pictures.
- Extremely thin and lightweight OLED displays can be constructed.
- The power consumption of OLED displays is extremely low.

OLEDs have some disadvantages compared to other displays:

- Manufacturing of OLED displays is costly.
- OLED displays have limited lifespans, usually 14 000 hours (corresponding to 5 years at 8 hours a day usage).
- OLED displays can be damaged by water and therefore tight sealing is required, which increases the cost.
- OLED displays suffer from screen burn-in problems, where pixels fade after displaying the same content for a long time.
- OLED displays can be damaged by exposure to UV light. As a result, OLED displays cannot be used in countries where the UV is very high. Manufacturers usually install UV blocking filters over the screen to protect the displays.
- The material used to produce blue light degrades more rapidly than the materials used for other colours. As a result, the colour balance of the overall display changes, causing the colours to be wrongly displayed.

1.4.4 LCD

The Liquid Crystal Display (LCD) is one of the most commonly used displays today. There are basically three types of LCDs as far as the type of data that can be displayed is concerned: Segment LCD, Dot Matrix LCD and Graphic LCD.

www.it-ebooks.info

Figure 1.6 16-segment LCD display

Segment LCD is also known as the alphanumeric LCD. These LCDs can display numbers represented by 7 segments, or numbers and Roman letters represented by 14 or 16 segments. In addition, symbols can also be displayed. The segment LCDs are limited to displaying numbers, text and symbols. If we need to display other characters or shapes, then either a dot matrix display or a graphic display should be used. Figure 1.6 shows a typical 16-segment LCD display.

Dot Matrix LCD is also known as the character LCD. The most commonly used dot matrix LCD displays are 2 lines of 16 characters. Each character is represented by 5×7 dots (or 5×8 characters including the cursor). Dot matrix LCDs can display alphanumeric data, including a subset of symbols. Figure 1.7 shows a typical dot matrix LCD display.

Graphic LCDs are composed of pixels and provide the greatest flexibility to the user. In a graphic LCD, pixels are arranged in rows and columns and each pixel can be addressed individually. Graphic LCDs are used when we need to display numbers, letters, symbols, shapes or pictures. Figure 1.8 shows a typical graphics LCD display.

LCD displays produce no light of their own and so require an external light source to be visible. On some displays, a cold cathode fluorescent lamp is inserted behind the LCD panel. On some other models, the ambient light is used to make the display visible.

Figure 1.7 Dot matrix LCD display

Figure 1.8 Graphics LCD display

www.it-ebooks.info

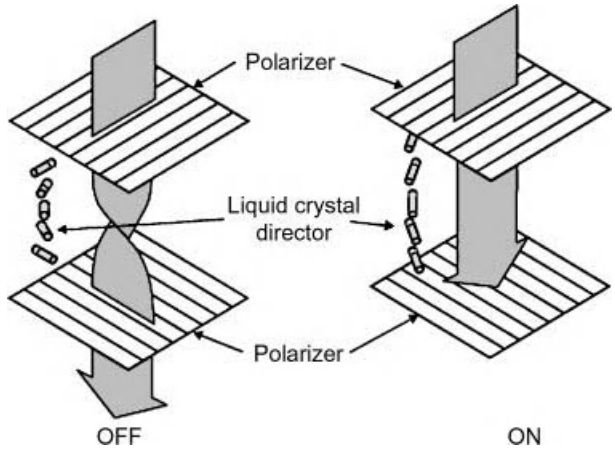


Figure 1.9 Operation of an LCD

LCD displays use the light modulating properties of liquid crystals. In a standard LCD display, a layer of molecules are aligned between two transparent Tin Oxide electrodes and two polarising filters placed at right angles to each other, as shown in Figure 1.9. Ambient light enters the LCD through the front polarising filter. The light then passes through the liquid crystals, which rotate the light passing through them. This rotation is usually 90 degrees in most type of LCDs. In the OFF state, since the light is rotated, it passes through the second polarising filter. Applying a voltage across the electrodes (ON state) orients the liquid crystals so that they are parallel to the electric field and the twisted structure disappears. Thus, the light is no longer rotated and light passing through the second polariser in the crossed shape is absorbed, thus causing the activated portion of the display to appear dark.

LCD displays can be classified as Passive Matrix and Active Matrix, depending upon the pixel addressing scheme used. A pixel matrix is addressed by rows and columns. In a passive display, transistors are used to activate rows and columns, not each pixel. In an active display, on the other hand, transistors are used at each red, green and blue pixel to keep the pixel at the desired intensity. In general, passive matrix displays are less costly and have narrower viewing angles than active matrix displays. Active matrix displays are sharper, have more contrast than passive displays, and also have faster response times.

There are many types of LCD displays, depending upon the amount and type of twisting used in liquid crystals. Some examples are: TN (Twisted Nematic -90° Twist), STN (Supertwisted Nematic -270° Twist), FSTN (Film Compensated STN), DSTN (Double Layer STN), and so on.

One of the LCD displays that has become popular recently is the (TFT (Thin Film Transistor) LCD, used in most mobile phones, laptop monitors and desktop computer monitors. TFT is an active matrix display providing the best resolution of all the LCD types, but it is also the most expensive type. In a TFT display, the transistors are made from a thin film of amorphous silicon deposited on a glass panel. TFT displays offer excellent response times, and sharp and crisp images. Some TFT displays are incorporated with touch screen hardware panels that enable the user to make a selection by touching the appropriate places on the screen.

www.it-ebooks.info

1.4.4.1 LCD Viewing Modes

Reflective: In this mode, LCDs use ambient light to illuminate the display and are therefore more suitable for outdoor use.

Transmissive: In this mode, LCDs use ambient light to illuminate the display and are therefore more suitable for indoor use.

Transflective: Transflective mode includes both reflective and transmissive types, so can be used both indoors and outdoors.

1.4.4.2 Key Specifications of LCDs

Resolution: Number of pixels, measured in horizontal and vertical (e.g. 1024 × 768);

View size: The diagonal size of the LCD display;

Dot pitch: The distance between two adjacent same colour pixels. The dot pitch is either specified as the number of dots per inch, or the distance is given in millimetres (e.g. 0.25 mm). The smaller the dot pitch size (or higher the number of dots per inch), the sharper the image becomes;

Response time: The minimum time it takes to change a pixel's brightness (or colour). The response time is measured in milliseconds and typical values are several milliseconds. A low response time is always desirable;

View angle: The angle from which the LCD can be viewed without loss of any detail;

Brightness: The amount of light emitted from the display;

Contrast ratio: The ratio of the luminance of the brightest colour (white) to that of the darkest colour (black). A high contrast ratio is a desirable feature of any display;

Aspect ratio: The ratio of the width of the LCD to its height (e.g. 16: 9, 4: 3, etc.)

1.5 Summary

This chapter has provided an introduction to the microcontroller and microprocessor based computer systems. The differences between the two types of computer systems have been explained in detail. In addition, some of the most commonly used concepts in microcontrollers have been described.

The final part of the chapter has provided an introduction to the display systems used in microcontroller based applications. An explanation of the important terms used in displays has also been given.

Exercises

1.1 What is a microprocessor?
1.2 What is a microcontroller? Explain the differences between a microprocessor and a microcontroller.
1.3 Where would you use a flash memory?
1.4 Where would you use RAM memory?
1.5 What is an A/D converter? Give an example for its use in a microcontroller based application.
1.6 What is the purpose of the watchdog module in a microcontroller?

www.it-ebooks.info

1.7 What happens when a microcontroller is reset?
1.8 How many types of LCDs are there? Which one would you choose if the number of I/O ports is limited?
1.9 What is a graphics LCD? Give an example for its use in practise.
1.10 Explain the operation of passive and active display technologies.
1.11 What is a TFT display? Why are TFT displays popular? In which type of applications are they commonly used?
1.12 What is an OLED display? Explain its operation. What are the differences between a TFT and an OLED display?
1.13 What are the advantages of touch screen displays? Give an example of touch screen based application.

www.it-ebooks.info

2

PIC18F Microcontrollers

PIC is a family of Harvard architecture microcontrollers (except the 32-bit devices) manufactured by Microchip Technology Inc. PIC microcontrollers are available in over 1000 models. Depending upon the data width used, we can classify these microcontrollers in three groups: 8-bit, 16-bit and 32-bit microcontrollers. Figure 2.1 shows an overview of the PIC series of microcontrollers.

The PIC 10, 12, 16 series are the low-end 8-bit microcontrollers with low speed, low pin count, low cost, small memories, with only 35 instructions, making them easy to learn and program. PIC18 series are medium-end 8-bit microcontrollers with medium speed, higher pin count, large memories, and having over 80 instructions. These microcontrollers include various on-chip modules, such as CAN, USB, SPI, multiple USARTs, several timers, multiplier hardware, and clock speeds up to 40 MHz. These microcontrollers are currently used in most new complex PIC microcontroller projects. PIC24 and dsPIC series are 16-bit high-speed microcontrollers with large memories and peripheral support, designed for time-critical applications where real-time processing is very important. These microcontrollers find applications in digital signal processing (DSP) and in high-speed automatic digital control systems. The architectures of these 16-bit microcontrollers are different to the 8-bit microcontrollers, as they are configured for high-speed processing required in DSP applications, having fast multiplication and addition modules (MACs).

The new PIC32 microcontroller family are 32-bit processors with standard Von Neumann architecture, having large memories and peripheral support, offering very high speed processing in highly precision applications. One of the advantages of PIC microcontrollers is that they support easy migration across product families. For example, a project designed using a PIC16 series microcontroller can easily be upgraded to use a PIC18 series microcontroller. This is especially true if the development was carried out using a high-level language such as C, which is compatible across all the 8-bit families.

Currently, most medium-speed general purpose projects with graphical display requirements are based on the PIC18F series, as they provide the required speed, large data and program memories, and large number of input-output capabilities.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

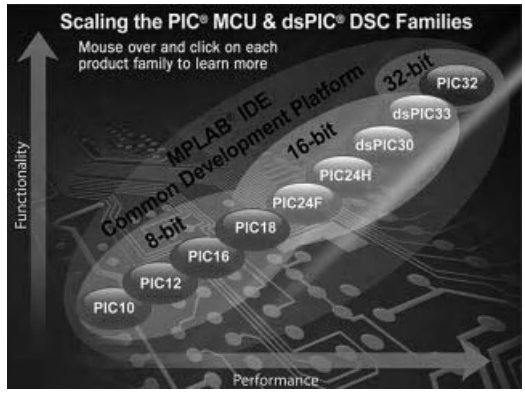


Figure 2.1 PIC microcontroller series

PIC18 microcontrollers are available in many models, from small 18-pin chips to 100-pin chips, program memories from 4 KB to 128 KB, data memories from 256 bytes to 4 KB, and input-output pins from 15 to 70.

In this chapter we look in detail at the architecture of a medium-end PIC18 microcontroller, namely the PIC18F2410, as it will be necessary to know the basic architecture of the PIC18 series of microcontrollers when we begin creating display based projects in later chapters. The reason for choosing the PIC18F2410 is because it is a low-cost, yet powerful microcontroller, having only 28 pins, and its architecture can be considered as representative of the PIC18F series.

2.1 The PIC18F2410 Microcontroller

The PIC18F2410 microcontroller belongs to the family PIC18F2X1X/4X1X. There are 8 microcontrollers in this family, with slightly different specifications. Table 2.1 gives the basic specifications of the microcontrollers in this family.

The basic features of the PIC18F2410 microcontroller are:

- 16 KB program memory;
- 768 bytes data memory;

Table 2.1 The PIX18F2X1X/4X1X microcontroller family

Device	Program memory	Data memory	Interrupt Sources	I/O	Timers	A/D converter	USART	Package
PIC18F2410	16 KB	768	18	25	4	10 ch	1	28
PIC18F2510	32 KB	1536	18	25	4	10 ch	1	28
PIC18F2515	48 KB	3968	18	25	4	10 ch	1	28
PIC18F2610	64 KB	3968	18	25	4	10 ch	1	28
PIC18F4410	16 KB	768	19	36	4	13 ch	1	40/44
PIC18F4510	32 KB	1536	19	36	4	13 ch	1	40/44
PIC18F4515	48 KB	3968	19	36	4	13 ch	1	40/44
PIC18F4610	64 KB	3968	19	36	4	13 ch	1	40/44

www.it-ebooks.info

- 25 I/O pins;
- each I/O pin has 25 mA source/sink capability;
- 10-bit 10 channel A/D converters;
- 18 interrupt sources;
- interrupt priority levels;
- 4 timers;
- DC to 40 MHz operating frequency;
- capture/compare/PWM modules;
- USART module;
- master synchronous serial port module (MSSP);
- low-voltage detection module (LVD);
- power-on reset (POR), power-up timer (PWRT), oscillator startup timer (OST);
- watchdog timer (WDT);
- 75 instructions (83 with extended instruction set enabled);
- 20 nA current consumption in sleep mode (CPU and peripherals off);
- 28-pin package.

2.2 PIC18F2410 Architecture

The pin configuration (DIP package) of the PIC18F2410 microcontroller is shown in Figure 2.2. As we shall see later, most of the pins are multiplexed and can be used for different purposes. For example, pin 2 is named as RA0/AN0 and this is the PORT A least significant port pin. This pin can be used as an analogue input (named AN0), or as a digital I/O (named RA0).

Figure 2.3 shows the simplified internal architecture of the PIC18F2410 microcontroller. The CPU is at the centre of the diagram and consists of an 8-bit ALU, an accumulator register (WREG), and an 8×8 multiplier module. The multiplier takes data from the accumulator register and the data bus, and provides the 16-bit result in registers PRODH and PRODL, where the result can be read through the data bus.

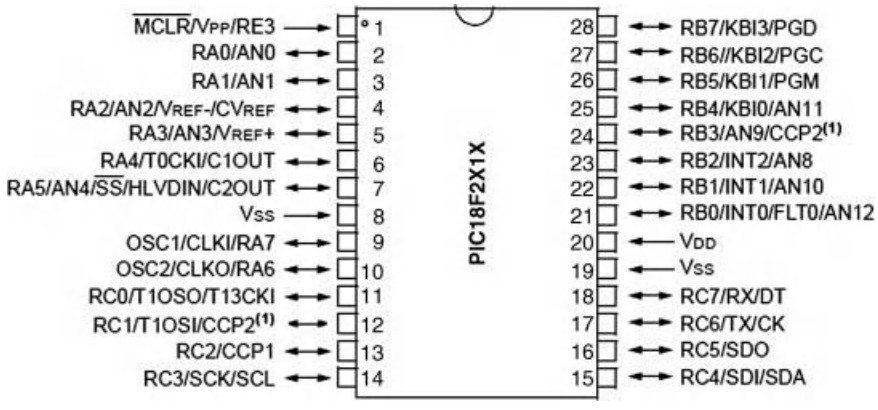


Figure 2.2 PIC18F2410 pin configuration (DIP package). (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

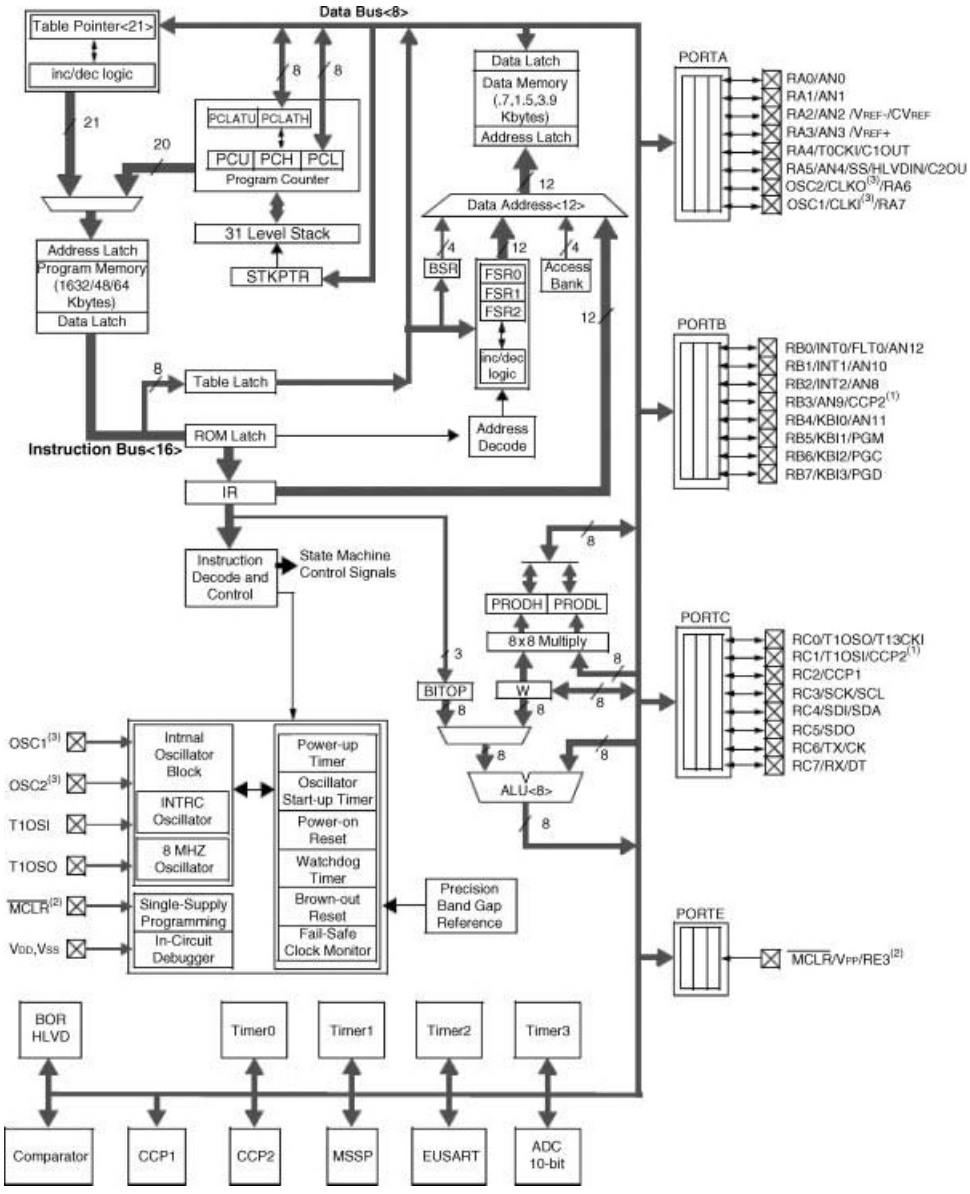


Figure 2.3 Internal architecture of the PIC18F2410 microcontroller. (Reproduced with permission from Microchip Inc)

The program memory and the program counter are shown at the top left corner of the figure. The memory address consists of 21 bits, capable of addressing up to 2 MB of memory data, although here only 16 KB is used. The program counter consists of two 8-bit registers PCH and PCL, and a 5-bit register PCU. A 32-level deep stack can be seen at the bottom of the program counter. The stack is used to store the return addresses when a subroutine is called or

www.it-ebooks.info

when an interrupt occurs. The stack is independent of the data memory and is addressed with a 5-bit stack pointer STKPTR. The stack pointer is initialised to 00 000 after a reset.

The data memory can be seen at the top right corner of the figure. The data addresses are 12 bits, thus up to 4 KB data can be addressed, although here only 768 bytes of data memory are implemented.

The instruction decode and control logic, located at the centre of the figure, decodes the instructions fetched from the program memory and sends the appropriate control signals to all parts of the microcontroller to implement the required operation.

Just below the instruction decode and control logic, we see the timing and power control module. This module is responsible for generating the clock timing pulses for both the external and internal clock. In addition, this module controls the power-on timer, oscillator startup, POR, watchdog timer, brown-out reset, single-supply programming, in-circuit debugger, and the fail safe clock monitoring.

At the bottom of the figure we can see the four timer modules, comparator/capture/pwm modules, master synchronous serial port module (MSSP), USART module, and the A/D converter module.

There are 4 I/O ports named PORTA, PORTB, PORTC and PORTE, and 25 I/O pins shown at the right side of the figure. PORTA, PORTB and PORTC are 8-bit ports, while PORTE has only 1 bit. All ports pins are bi-directional when configured as digital I/O.

2.2.1 The Program Memory

Figure 2.4 shows a memory map of the PIC18F2410 microcontroller. The device has a 21-bit program counter (PC <20: 0>), capable of addressing up to 2 MB of memory, although here only 16 KB is used, ranging from 00 000 h to 03 FFFh. Memory addresses above 0400 h are read as 0 and are not available. The Reset vector is at address 00 000 h and the program counter is loaded with this address after a reset, causing the program starting at this address to be executed. Addresses 00 008 h and 00 018 h are the high and low priority interrupt vectors, respectively. Thus, for example, when a low priority interrupt occurs, the program jumps to address 00 008 h.

An instruction cycle in an 8-bit PIC microcontroller consists of 4 cycles (Q1 to Q4). A fetch cycle begins with the program counter incrementing in Q1. The fetched instruction is decoded and executed in cycles Q2, Q3 and Q4. A data memory location is read during the Q2 cycle and written during the Q4 cycle. Because an instruction cycle consists of 4 cycles, the performance of a PIC microcontroller is measured by dividing the operating clock frequency by 4. For example, a processor operating with a 40 MHz clock frequency has a MIPS (Million Instructions Per Second) rating of 10 MIPS.

2.2.2 The Data Memory

Figure 2.5 shows the data memory of the PIC18F2410 microcontroller. Data memory is addressed with 12 bits, capable of addressing up to 4 KB of memory. The memory is usually divided into 16 banks, each bank 256 bytes long. The PIC18F2410 microcontroller uses only the first 3 banks (BANK 0, BANK 1 and BANK 2) from address 000 h to 2 FFh. The remaining banks, except half of BANK 15, are not used and return 0 when accessed. The upper part

www.it-ebooks.info

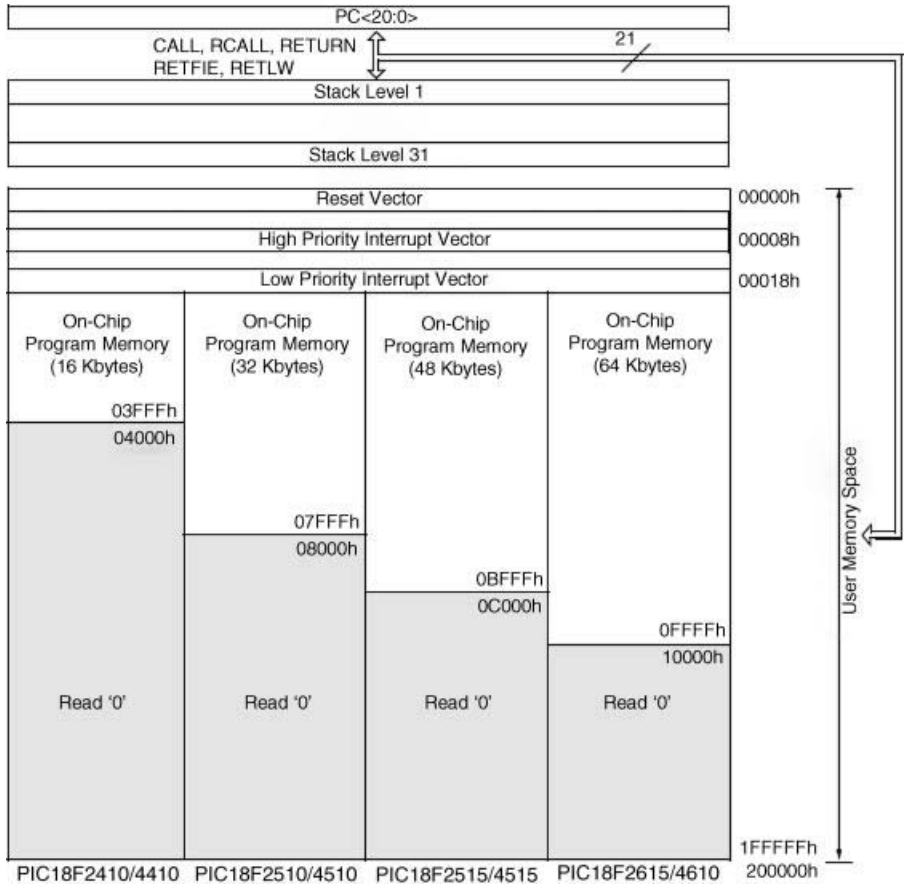


Figure 2.4 Program memory map of the PIC18F2410 microcontroller. (Reproduced with permission from Microchip Inc)

of BANK 15 is reserved for the SFR (Special Function Registers) registers. SFR registers control internal modules of the microcontrollers, such as A/D converter, interrupts, timers, USART, I/O ports, and so on.

2.2.3 Power Supply Requirements

The PIC18F2410 microcontroller operates with a power supply of 4.2 to 5.5 V, at the full speed of 40 MHz. The low-power version of the microcontroller (PIC18LF2410) can operate at a voltage as low as 2.0 V. As shown in Figure 2.6, at low voltages the maximum operating frequency is limited. For example, at 2.0 V the maximum operating frequency should not exceed 4 MHz. In practical applications, most microcontrollers are operated with a supply of 5.0 V, derived using a 78L05/7805 or a similar voltage regulator.

www.it-ebooks.info

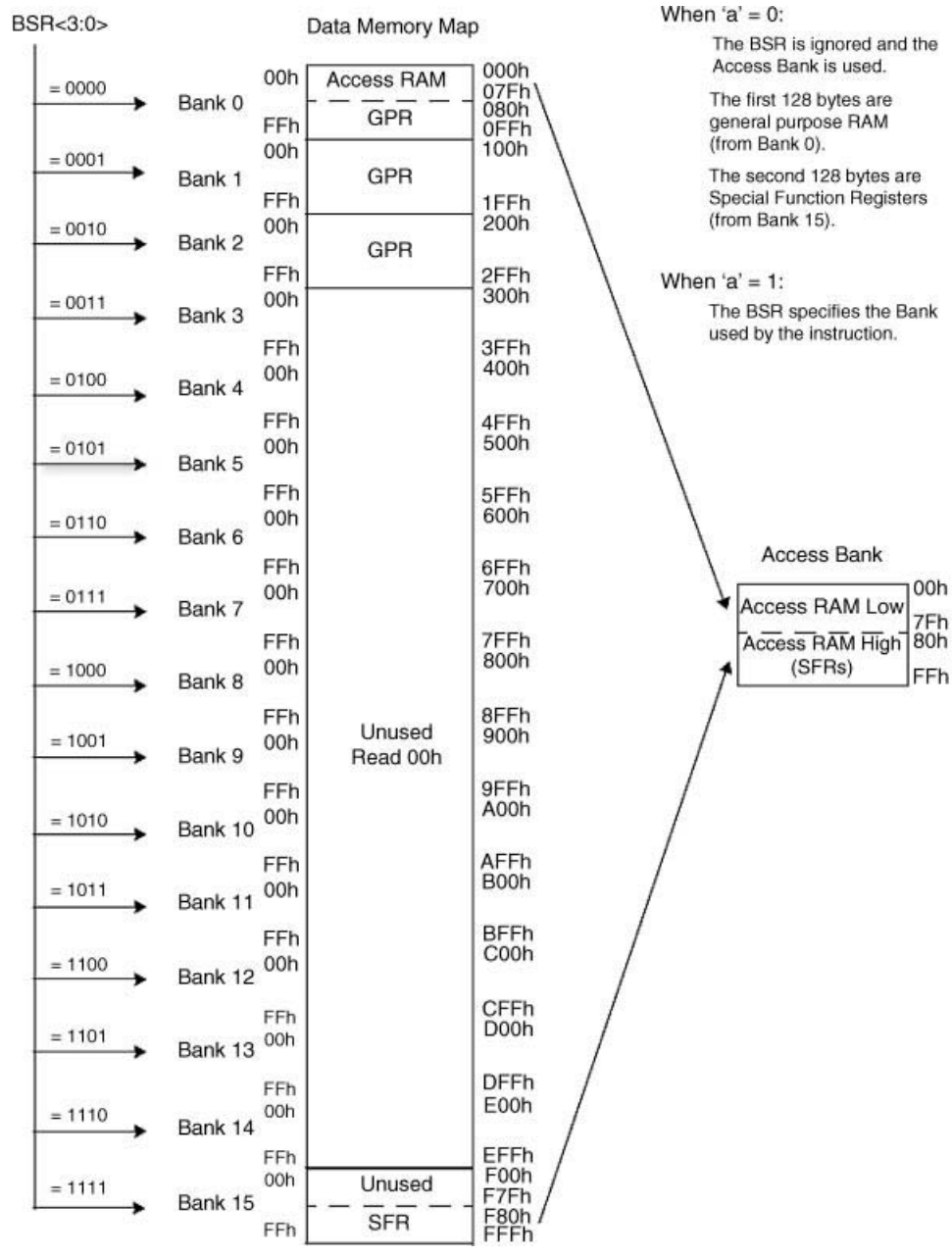


Figure 2.5 Data memory map of the PIC18F2410 microcontroller. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

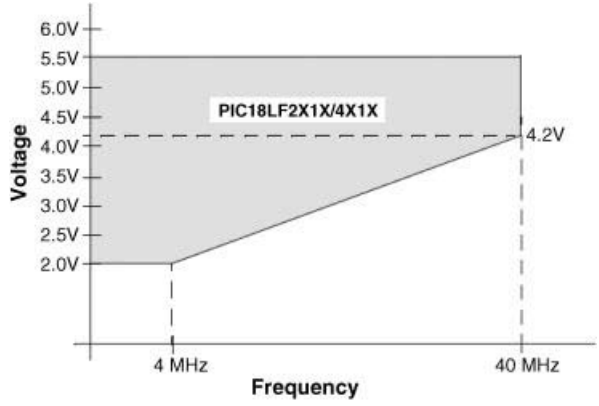


Figure 2.6 Power supply requirements. (Reproduced with permission from Microchip Inc)

2.2.4 Oscillator Configurations

The PIC18F2410 microcontroller can be operated in 10 different oscillator modes. The user can program the required oscillator mode during programming of the device. These modes are:

- low-power crystal (LP);
- Crystal/Resonator (XT);
- high-speed crystal/resonator with PLL enabled (HSPLL);
- external resistor-capacitor with Fosc/4 on OSC2 (RC);
- external resistor-capacitor with I/O on OSC2 (RCIO);
- internal oscillator with Fosc/4 on OSC2 and I/O on OSC1 (INTIO1);
- internal oscillator with I/O on OSC2 and OSC1 (INTIO2);
- external clock with Fosc/4 output on OSC2(EC);
- external clock with I/O on OSC2 (ECIO).

2.2.4.1 Using Crystal (LP/XT)

A crystal should be used in applications requiring high timing accuracies. The crystal is connected to pins OSC1 and OSC2 of the microcontroller with a pair of capacitors, as shown in Figure 2.7. That capacitor value depends on the oscillator mode and is shown in Table 2.2. In most applications, a 15–33 pF capacitor should be sufficient to achieve stability.

2.2.4.2 Using Resonator (XT)

A resonator should be used in low-cost applications where high timing accuracy is not essential. Resonators are available at low to medium frequencies, up to 10 MHz. The resonator should be connected to pins OSC1 and OSC2 of the microcontroller, as shown in Figure 2.8.

www.it-ebooks.info

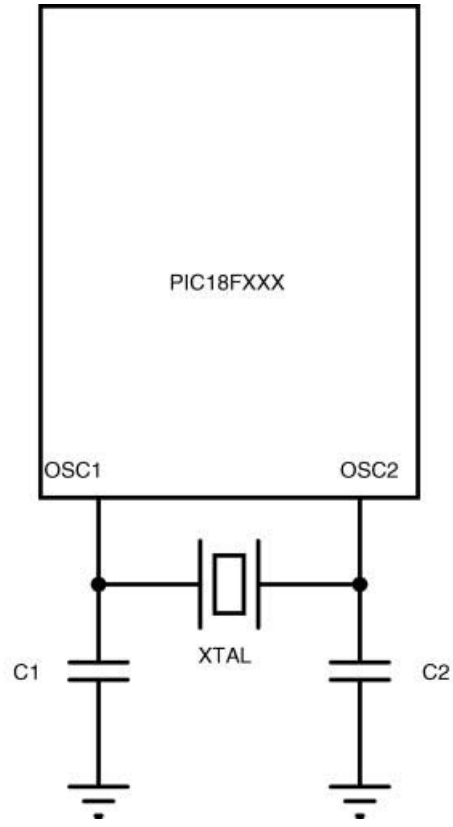


Figure 2.7 Operation with a crystal

Table 2.2 Required capacitor values

Mode	Frequency	C1,C2 (pF)
LP	32 kHz	33
	200 kHz	15
XT	200 kHz	22–68
	1.0 MHz	15
	4.0 MHz	15
HS	4.0 MHz	15
	8.0 MHz	15–33
	20.0 MHz	15–33
	25.0 MHz	15–33

2.2.4.3 Using External Resistor-Capacitor (RC/RCIO)

Using an external resistor-capacitor (RC) for timing provides the cheapest solution. Here, the clock frequency depends on the chosen resistor and capacitor values, component tolerances,

www.it-ebooks.info

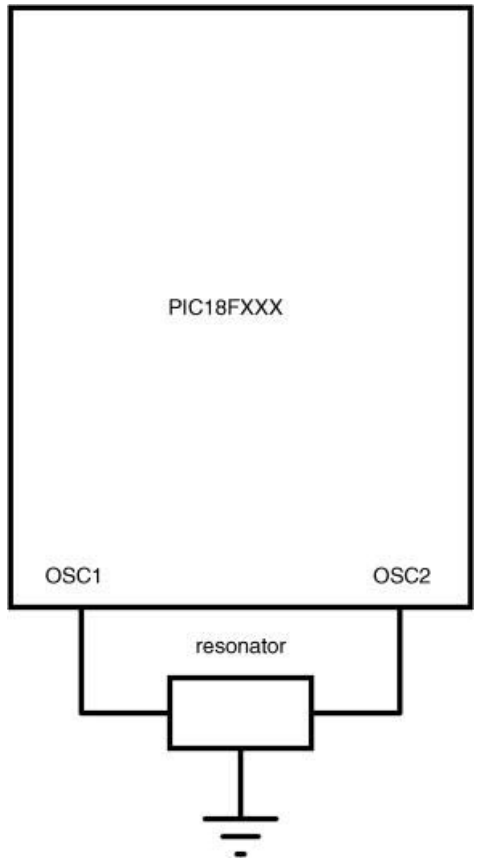


Figure 2.8 Operation with a resonator

power supply, temperature, and ageing of components. The clock frequency is not accurate and can easily change from unit to unit due to component tolerances.

Table 2.3 gives the approximate clock frequency with different RC combinations. The resistor should be between 3 K and 100 K, and the capacitor should be greater than 20 pF.

Table 2.3 Selecting a resistor and capacitor

C (pF)	R (K)	Frequency (MHz)
22	3.3	3.3
	4.7	2.3
	10.0	1.08
30	3.3	2.4
	4.7	1.7
	10.0	0.793

www.it-ebooks.info

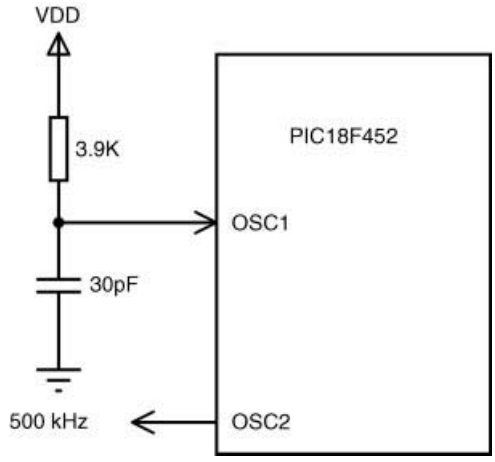


Figure 2.9 Operation with a resistor-capacitor

The clock frequency is given approximately by

$$f = 1/(4.2\,RC) \tag{2.1}$$

As an example, for a 2 MHz clock, we can choose a capacitor of 30 pF and a resistor of 3.9 K. Figure 2.9 shows the circuit diagram for RC mode operating at approximately 2 MHz. Notice that in RC mode, a clock is output from pin OSC2 with a frequency of Fosc/4, that is 500 Hz in this example.

The RCIO mode is similar to the RC mode, except that in RCIO mode the OSC2 pin is available as a general purpose I/O.

2.2.4.4 Using the Internal Oscillator (INTIO1/INTIO2)

An internal oscillator can be extremely useful in many applications. First, it eliminates the need to use an external timing device, thus reducing the cost and the component count. Second, by using an internal oscillator, the microcontroller oscillator pins become available for general purpose I/O.

PIC18F2410 includes two internal oscillators. A factory calibrated 8 MHz clock source (IINTOSC), and an RC based 31 kHz clock source (INTRC). In the INTIO1 mode, the OSC2 pin outputs a clock at frequency Fosc/4, while the OSC1 pin (RA7) can be used as general digital I/O. In the INTIO2 mode, both OSC1 and OSC2 pins function as general purpose I/O pins (RA6 and RA7).

Although the 8 MHz clock source is factory calibrated, the frequency can drift slightly and the SFR register OSCTUNE can be used to re-calibrate this clock source (see manufacturers' data sheet for more information). The 8 MHz clock drives a postscaler, and a multiplexer is used to provide clock frequencies in the range 31 kHz to 8 MHz.

Figure 2.10 shows the internal structure of the clock selection mechanism. SFR register OSCCON controls the clock selection, as shown in Figure 2.11. For example, to select an 4 MHz internal clock, bits <6: 4> of OSSCON should be set to binary pattern '110'.

www.it-ebooks.info

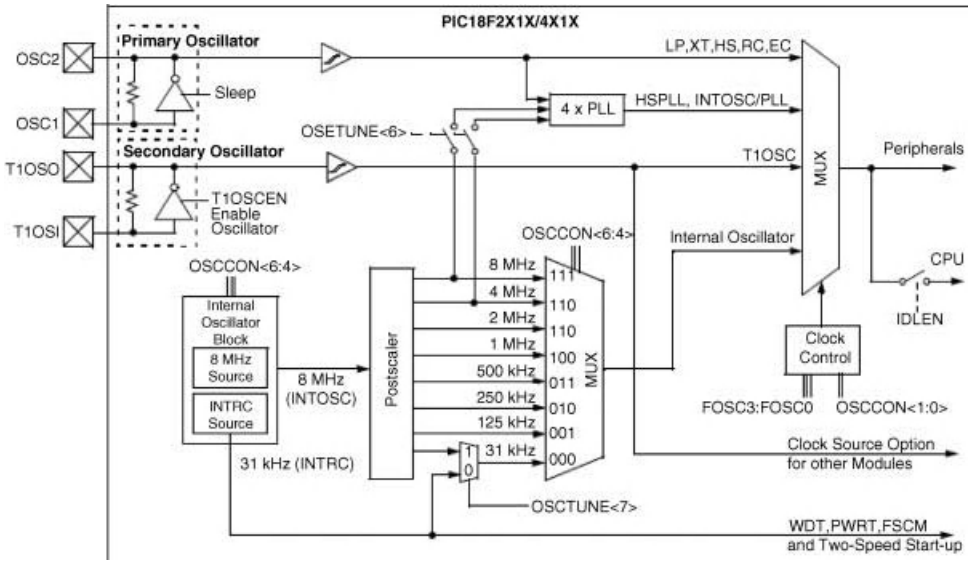


Figure 2.10 Clock selection mechanism. (Reproduced with permission from Microchip Inc)

The PIC18F2410 microcontroller includes a feature that allows the device clock source to be switched from the main oscillator to an alternative lower frequency clock source for conserving power. Essentially, there are two clock sources: Primary oscillator and Secondary oscillator. The primary oscillator includes the external crystal and resonator modes, external RC modes, and the internal oscillator. The Secondary oscillator is where Timer 1 is used as an internal oscillator.

2.2.4.5 Using External Clock (EC/ECIO)

In this mode of operation, an external clock source is connected to pin OSC1 of the microcontroller. In EC mode, pin OSC2 (RA6) provides a clock output at the frequency Fosc/4. In ECIO mode, pin OSC2 (RA6) is available as a general purpose I/O. Figure 2.12 shows operation with an external clock.

2.2.4.6 High-speed Crystal/resonator with PLL (HSPLL)

High-speed operation is possible using the Phase Locked Loop (PPL) to multiply the selected clock source by 4. The PLL operation is available either with the external crystal/resonator (HSPLL), or by using the internal clock.

In the HSPLL mode of operation, an external crystal or resonator is connected to the OSC1 and OSC2 pins, as in mode XT. The internal PLL of the microcontroller is then programmed to multiply the clock frequency by 4 to provide higher frequencies. External crystal/resonator up to 10 MHz can be used in this mode, to provide an operating frequency of 40 MHz. Bits FOSC3:FOSC0 of the Configuration register 1H are used to set the HSPLL mode (see manufacturers' data sheet for more information).

www.it-ebooks.info

R/W–0	R/W–1	R/W–0	R/W–0	R(1)	R–0	R/W–0	R/W–0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0
bit 7							bit 0

bit 7 **IDLEN:** Idle Enable bit
1 = Device enters Idle mode on SLEEP instruction
0 = Device enters Sleep mode on SLEEP instruction

bit 6–4 **IRCF2:IRCF0:** Internal Oscillator Frequency Select bits
111 = 8 MHz (INTOSC drives clock directly)
110 = 4 MHz
101 = 2 MHz
100 = 1 MHz(3)
011 = 500 kHz
010 = 250 kHz
001 = 125 kHz
000 = 31 kHz (from either INTOSC/256 or INTRC directly)(2)

bit 3 **OSTS:** Oscillator Start-up Time-out Status bit(1)
1 = Oscillator start-up time-out timer has expired; primary oscillator is running
0 = Oscillator start-up time-out timer is running; primary oscillator is not ready

bit 2 **IOFS:** INTOSC Frequency Stable bit
1 = INTOSC Frequency is Stable
0 = INTOSC Frequency is not Stable

bit 1–0 **SCS1: SCS0:** System Clock Select bits
1x = Internal oscillator block
01 = Secondary (Timer1) oscillator
00 = Primary oscillator

Note 1: Reset state depends on state of the IESO Configuration bit.
2: Source selected by the INTSRC bit (OSCTUNE<7>), see text.
3: Default output frequency of INTOSC on Reset.

Figure 2.11 OSCCON register bit definitions. (Reproduced with permission from Microchip Inc)

When used with the internal clock sources, the PLL can produce clock speeds of 16 MHz (with a 4 MHz internal clock) or 32 MHz (with an 8 MHz internal clock). The PLL for the internal clock sources is enabled by setting bit 6 of the SFR register OSCTUNE, as shown in Figure 2.13.

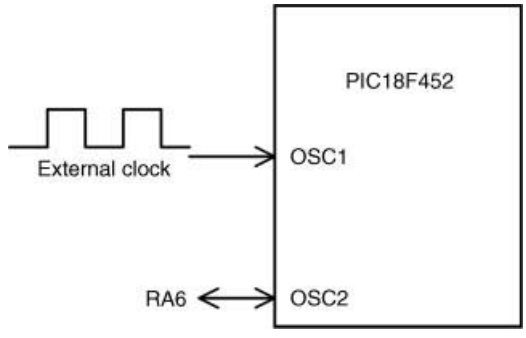


Figure 2.12 Operation with an external clock

www.it-ebooks.info

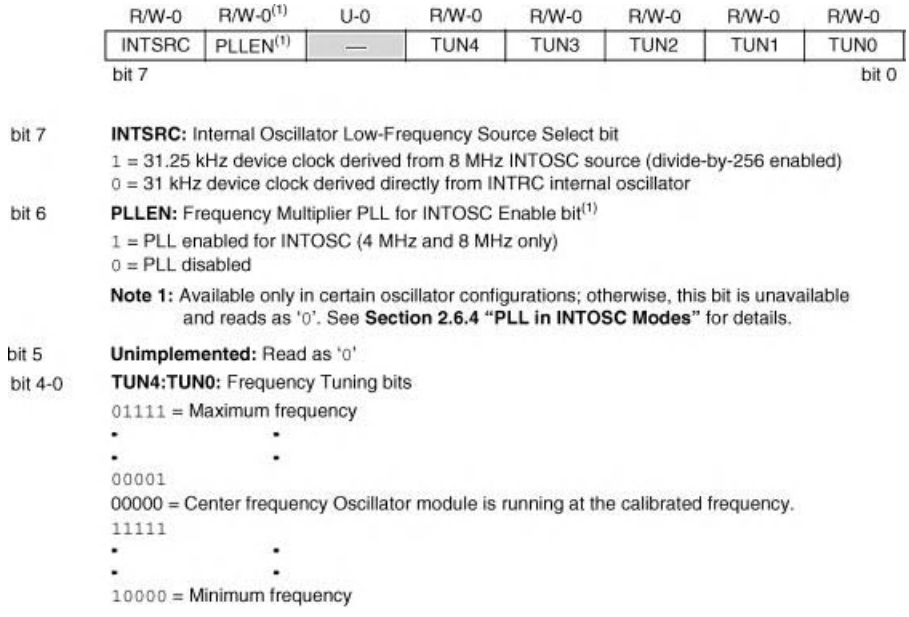

R/W-0 R/W-0(1) U-0 R/W-0 R/W-0 R/W-0 R/W-0 R/W-0
INTSRC PLLEN(1) — TUN4 TUN3 TUN2 TUN1 TUN0
bit 7 bit 0
bit 7 INTSRC: Internal Oscillator Low-Frequency Source Select bit
1 = 31.25 kHz device clock derived from 8 MHz INTOSC source (divide-by-256 enabled)
0 = 31 kHz device clock derived directly from INTRC internal oscillator
bit 6 PLLEN: Frequency Multiplier PLL for INTOSC Enable bit(1)
1 = PLL enabled for INTOSC (4 MHz and 8 MHz only)
0 = PLL disabled
Note 1: Available only in certain oscillator configurations; otherwise, this bit is unavailable and reads as '0'. See Section 2.6.4 "PLL in INTOSC Modes" for details.
bit 5 Unimplemented: Read as '0'
bit 4-0 TUN4:TUN0: Frequency Tuning bits
01111 = Maximum frequency
00001
00000 = Center frequency Oscillator module is running at the calibrated frequency.
11111
10000 = Minimum frequency

Figure 2.13 OSCTUNE register bit definitions. (Reproduced with permission from Microchip Inc)

2.2.5 The Reset

The Reset action puts the microcontroller into a known state, where the program counter is loaded with address 0 and program execution starts from this address. There are several actions that may cause a reset action:

- Power-on Reset (POR);
- MCLR Reset during normal operation;
- MCLR Reset during power-managed modes;
- Watchdog Timer (WDT) Reset;
- Programmable Brown-out Reset (BOR);
- Reset instruction;
- Stack full Reset;
- Stack underflow Reset.

In this section we are interested in the most commonly used reset actions: Power-on Reset and MCLR Reset during normal operation.

2.2.5.1 Power-on Reset

A Power-on Reset (POR) is generated when power is applied to the microcontroller and when the supply voltage rises above a certain threshold. During POR, internal parts of the microcontroller are initialised. The MCLR pin should be connected to the supply voltage

www.it-ebooks.info

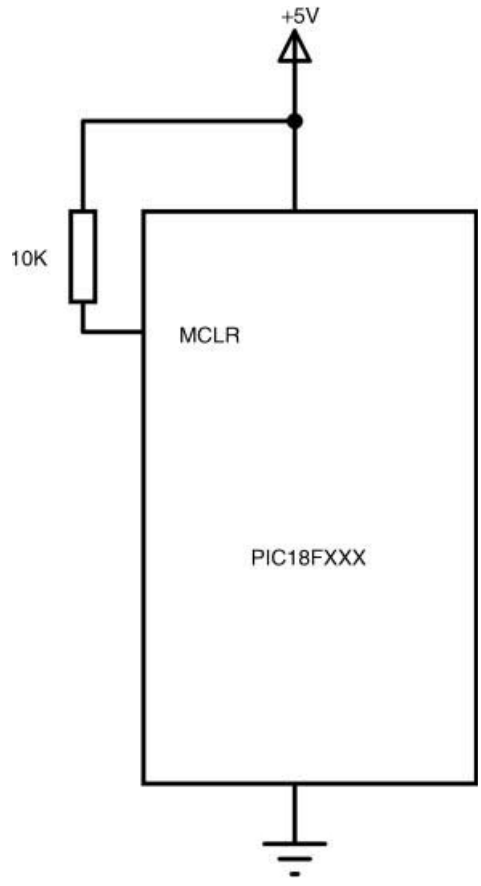


Figure 2.14 Power-on Reset circuit

with a 1 K to 10 K resistor before the POR is enabled. Figure 2.14 shows a typical POR circuit, where the MCLR pin is connected to the supply voltage via a 10 K resistor. In applications where the rise time of the supply voltage is small, it is recommended to use a diode and a capacitor in the POR circuit, as shown in Figure 2.15.

2.2.5.2 External Reset

In some applications we may want to force an external reset action, for example by pressing a button. This can easily be arranged by the circuit shown in Figure 2.16. The MCLR pin is normally at logic HIGH during normal operation. Pressing the button forces MCLR to be LOW, which causes a reset action.

2.2.6 Parallel I/O Ports

The PIC18F2410 microcontroller supports 4 I/O ports named PORT A, PORT B, PORTC and PORT E. The first three ports are 8-bits wide, while PORT E has only 1 bit. The bits of a

www.it-ebooks.info

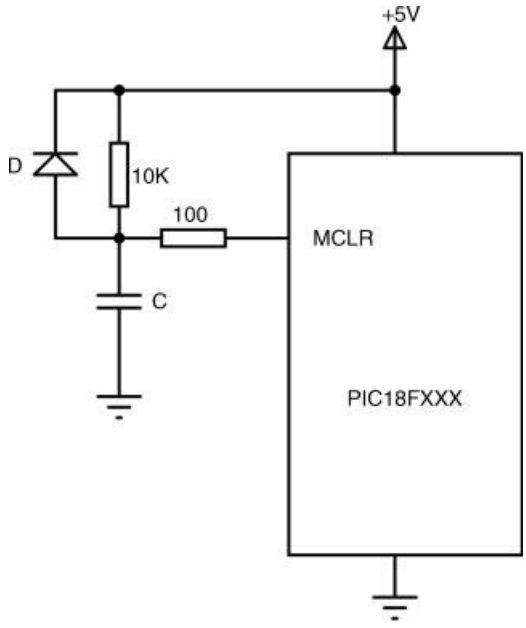


Figure 2.15 Power-on circuit for slow rising supply voltage

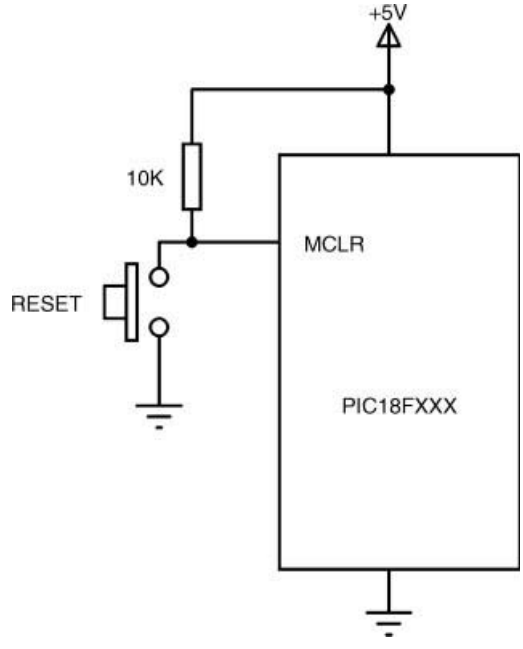
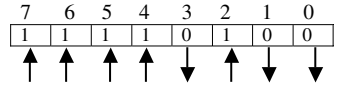


Figure 2.16 External Reset circuit

www.it-ebooks.info

port are named as RPn, where P is the port name, and n is the bit number. For example, RB0 is bit 0 of PORT B. Similarly, RA7 is bit 7 of PORT A, and so on.

The I/O ports are bi-directional. An input port pin can easily be changed to become an output pin, and vice versa. Port pin directions must be configured before they are used. The SFR register TRIS is used to configure the port directions. Each port register has a corresponding TRIS register. Thus, for example, the TRIS register for PORT A is TRISA, the TRIS register for PORT B is TRISB, and so on. Clearing a bit in a TRIS register forces the corresponding port pin to become an output. Similarly, setting a bit in a TRIS register forces the corresponding port pin to become an input. For example, to configure pins 0, 1, 3 of PORT B to become output and the remaining pins to become input, we have to load the following values into the TRISB register:



TRISB = 0xF4;

In addition to the standard port registers, every port has a latch register. This register is called LATx, where x is the port name. For example, PORT A latch register is LATA, PORT B latch register is LATB, and so on. The latch register holds the actual value sent to a port pin. Thus, for example, when reading from a port pin, we have two choices. If we read from the latch register, then the value read is unaffected by any external device connected to the port pin. If, on the other hand, the port pin is pulled low (even though a logic HIGH was sent to the port) by an externl device, then reading the port register will give a HIGH value.

2.2.6.1 PORT A

PORT A is an 8-bits wide, parallel I/O port, with the pin configuration shown in Table 2.4. As shown, the port pins are multiplexed and can be used for different purposes. Most of the PORT A pins can be configured either as digital I/O or as analogue inputs. This port has the following registers associated with it:

- PORTA register.
- TRISA register.
- LATA register.

Register PORTA is used to write and read data from the port pins. TRISA register is used to set the port pin directions. LATA is the latch register used to read a latched output value from the port.

Figure 2.17 shows the simplified internal block diagram of a generic port pin without the peripheral functions. The port pin consists of three D-type latch registers and a number of buffers. The output and input operations are described:

www.it-ebooks.info

Table 2.4 PIC18F2410 microcontroller PORT A pin functions

Pin	Description
RA0/AN0	
RA0	Digital I/O
AN0	Analogue input 0
RA1/AN1	
RA1	Digital I/O
AN1	Analogue input 1
RA2/AN2/VREF-/CVREF	
RA2	Digital I/O
AN2	Analogue input 2
VREF−	A/D reference voltage (low) input
CVREF	Comparator reference output
RA3/AN3/VREF+	
RA3	Digital I/O
AN3	Analogue input 3
VREF+	A/D reference voltage (high) input
RA4/T0CKI/C1OUT	
RA4	Digital I/O
T0CKI	Timer 0 external clock input
C1OUT	Comparator 1 output
RA5/AN4/SS/HLVDIN/C2OUT	
RA5	Digital I/O
AN4	Analogue input 4
SS	SPI Slave Select input
HLVDIN	High/Low voltage detect input
C2OUT	Comparator 2 output
RA6/OSC2/CLKO	Digital I/O
RA6	Digital I/O
OSC2	Oscillator input
CLKO	Clock output
RA7/OSC1/CLKI	Digital I/O
RA7	Digital I/O
OSC1	Oscillator input
CLKI	Clock input

Output: TRIS register is loaded with logic 0, thus the Q output of TRIS Latch is 0, enabling the output buffer to the I/O pin. If a data is now placed on the D input of the data latch, this data appears as the output of the port pin.

Input using port input: TRIS register is loaded with logic 1, thus the Q output of TRIS Latch is 1, disabling the output buffer to the I/O pin. To read data from the I/O pin, the RD Port pin is set to logic 1, thus enabling the input latch and placing the input data on the data bus.

Input using LATx register: TRIS register is loaded with logic 1, thus Q output of TRIS Latch is 1, disabling the output buffer to the I/O pin. Setting pin RD LAT to 1 reads the data

www.it-ebooks.info

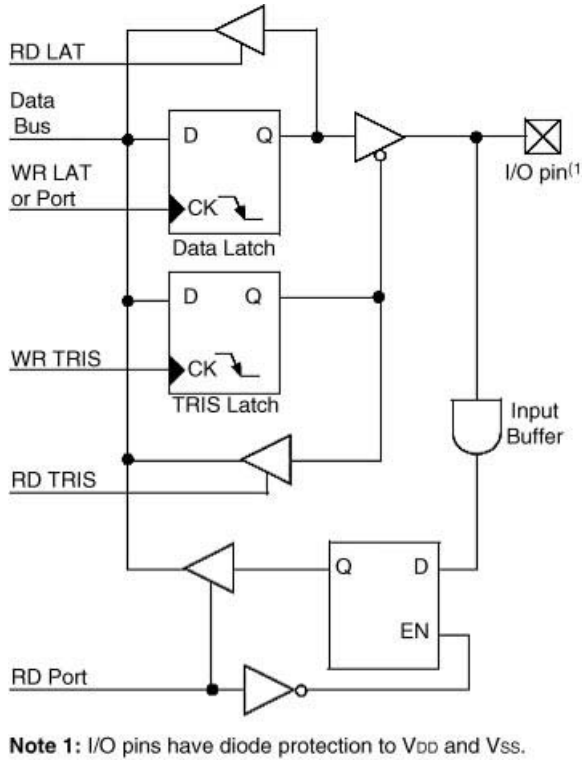


Figure 2.17 Generic port pin without peripheral functions. (Reproduced with permission from Microchip Inc)

at the output of the data latch and places it on the data bus. Remember that the data read here is the actual data sent to the port output earlier, and this data is not affected by any devices connected to the I/O pin.

On POR, PORT A pins RA5 and RA3:RA0 are configured as analogue inputs. Pin RA4 is configured as digital input. SFR register ADCON1 can be used to change the port configuration. For example, setting ADCON1 to $0 \times 7F$ configures all PORT A pins to become digital.

2.2.6.2 PORT B

PORT B is an 8-bit parallel port with the pin configuration shown in Table 2.5. As with PORT A, all the PORT B pins are multiplexed with other functions, such as analogue inputs and interrupt inputs.

As in PORT A, PORT B has the following registers associated with it:

- PORTB register.
- TRISB register.
- LATB register.

www.it-ebooks.info

Table 2.5 PIC18F2410 microcontroller PORT B pin functions

Pin	Description
RB0/INT0/FLT0/AN12	
RB0	Digital I/O
INT0	External interrupt 0
FLT0	PWM fault input for CCP1
AN12	Analogue input 12
RB1/INT1/AN10	
RB1	Digital I/O
INT1	External interrupt 1
AN10	Analogue input 10
RB2/INT2/AN8	
RB2	Digital I/O
INT2	External interrupt 2
AN8	Analogue input 8
RB3/AN9/CCP2	
RB3	Digital I/O
AN9	Analogue input 9
CCP2	Capture 2 input, Compare 2 and PWM2 output
RB4/KBIO/AN11	
RB4	Digital I/O
KBIO1	Interrupt on change input
AN11	Analogue input 11
RB5/KBI1/PGM	
RB5	Digital I/O, Interrupt on change pin
KBI1	Interrupt on change input
PGM	Low voltage ICSP programming pin
RB6/KBI2/PGC	
RB6	Digital I/O, Interrupt on change pin
KBI2	Interrupt on change input
PGC	In-circuit debugger and ICSP programming pin
RB7/KBI3/PGD	
RB7	Digital I/O, Interrupt on change pin
KBI3	Interrupt on change input
PGD	In-circuit debugger and ICSP programming pin

On POR, PORT B pins RB4:RB0 are configured as analogue inputs and RB7:RB5 pins are configured as digital inputs. Configuration register PBADEN or SFR register ADCON1 can be programmed to change the PORT B pin configuration. For example, when PBADEN is set to 1, all pins with analogue functions are set to analogue input mode.

2.2.6.3 PORT C

PORT C is also an 8-bit bi-directional port with multiplexed pins. The pin configuration of PORT C is shown in Table 2.6. Most points of PORT C are multiplexed with timer, USART and SPI bus functions.

www.it-ebooks.info

Table 2.6 PIC18F2410 microcontroller PORT C pin functions

Pin	Description
RC0/T1OSO/T13CKI	
RC0	Digital I/O
T1OSO	Timer 1 oscillator output
T13CKI	Timer 1/Timer 3 external clock input
RC1/T1OSI/CCP2	
RC1	Digital I/O
T1OSI	Timer 1 oscillator input
CCP2	Capture/compare/PWM 2 output
RC2/CCP1	
RC2	Digital I/O
CCP1	Capture/compare/PWM 1 output
RC3/SCK/SCL	
RC3	Digital I/O
SCK	Clock for SPI mode
SCL	Clock for I2C mode
RC4/SDI/SDA	
RB4	Digital I/O
SDI	SPI data in
SDA	I2C data I/O
RC5/SDO	
RC5	Digital I/O
SDO	SPI data out
RC6/TX/CK	
RC6	Digital I/O
TX	USART transmit
CK	USART synchronous clock
RC7/RX/DT	
RC7	Digital I/O
RX	USART receive
DT	USART synchronous data

As in the other ports, PORT C has the following registers associated with it:

- PORT C register.
- TRISC register.
- LATC register.

On power-on, all pins of PORT C are configured as digital inputs.

2.2.6.4 PORT E

On the PIC18F2410 microcontroller, PORT E is a single bit input only port, multiplexed with the MCLR (reset pin) and VPP (programming voltage pin) functions, and is available when MCLRE = 0.

www.it-ebooks.info

2.2.7 Timer Modules

PIC18F2410 microcontroller includes 4 timer modules, TIMER 0, TIMER 1, TIMER 2 and TIMER 3. The structure and the operation of each timer are briefly described in this section.

2.2.7.1 TIMER 0

TIMER 0 operates in both 8- and 16-bit modes. The timer can be clocked from external or internal clock sources. An 8-bit programmable prescaler is provided to divide the clock frequency. An *interrupt on overflow* event can be declared so that an interrupt is generated whenever the timer overflows.

2.2.7.1.1 8-bit Mode

Figure 2.18 shows TIMER 0 block diagram when operating in 8-bit mode. The external clock input is the T0CKI pin. The internal clock is derived by dividing the oscillator frequency by 4.

Operation of TIMER 0 is controlled by SFR register T0CON. Figure 2.19 shows the bit definitions of T0CON.

The prescaler value can be selected between 2 and 256, using bits T0PS2:T0PS0 of T0CON. TMR0L is the timer register when operating in 8-bit mode. The register counts up at every clock pulse and overflows when the count changes from 255 to 0. A timer interrupt is generated after an overflow, if the timer interrupt is enabled.

The time to overflow is calculated using the following equation:

$$\text{Time to overflow} = 4 * \text{Clock period} * \text{Prescaler} * (256 - \text{TMR 0L}) \tag{2.2}$$

where

time to overflow is in μs;
clock period is in μs;
TMR0L is the initial value loaded (0 to 255) into register TMR0L.

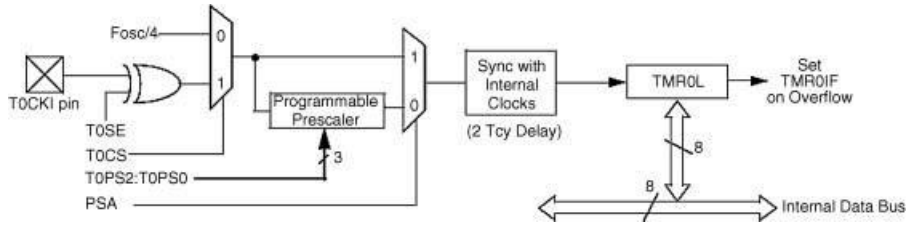


Figure 2.18 TIMER 0 block diagram in 8-bit operation. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT:** Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA:** Timer0 Prescaler Assignment bit
1 = Timer0 Prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 Prescaler is assigned.Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0:** Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Figure 2.19 T0CON bit definitions. (Reproduced with permission from Microchip Inc)

For example, if we assume an 8 MHz clock, and the prescaler is chosen as 16 by setting bits PS2:PS0 to 011, and also assume that the timer register is loaded with decimal 100, the time to overflow will be given by:

```
Clock period is T = 1/f = 1/8 = 0.125 µs
Time to overflow = 4 * 0.125 * 16 * (256 - 100) = 1248 µs
```

Thus, the timer will overflow after 1.248 ms.

In most applications we want to know what value to load into TMR0L for a required time to overflow. Re-arranging the above equation, we get

$$\text{TMR0L} = 256 - (\text{Time to overflow})/(4 * \text{Clock period} * \text{Prescaler}) \qquad (2.3)$$

An example is given below.

Example 2.1

It is required to generate a timer overflow after 500 µs using TIMER 0. Assuming that the clock frequency is 8 MHz, and the prescaler value is 16, calculate the value to be loaded into timer registers.

www.it-ebooks.info

Solution 2.1
Using the above formula we get

$$\text{TMR0L} = 256 - (500)/(4 * 0.125 * 16) = 193.5 \tag{2.4}$$

The closest integer is 193. Thus, the timer registers will be loaded as follows:

```
TMR0L = 193              // Timer value
T0CS = 0;                // Select internal clock
T0PS2 = 0;               // Set prescaler to 16
T0PS1 = 1;               // ..
T0PS0 = 1;               // ..
PSA = 0;                 // Select prescaler
```

2.2.7.1.2 16-bit Mode

Figure 2.20 shows the TIMER 0 block diagram when operating in 16-bit mode. In this mode there are two timer registers, TMR0H and TMR0L. The timer high byte is not directly accessible and is updated with the help of register TMR0H. All 16 bits of the timer is updated at once. To read the 16-bit timer value, we first read the low byte TMR0L, and during this operation the high byte is loaded into register TMR0H, which can be read in a second read operation. Similarly, to write a 16-bit value to the timer register, we first write the high byte into TMR0H. Then, when the low byte is written to TMR0L, both TMR0L and TMR0H are written to the timer.

2.2.7.2 TIMER 1

TIMER 1 is a 16-bit timer that can be operated as either a timer or a counter. SFR register T1CON controls TIMER 1, as shown in Figure 2.21. When operated as a timer, TMR1CS bit of T1CON is low, thus selecting the internal oscillator (Fosc/4). When TMR1CS is high, external clock from pin T1OSO is selected as the clock source.

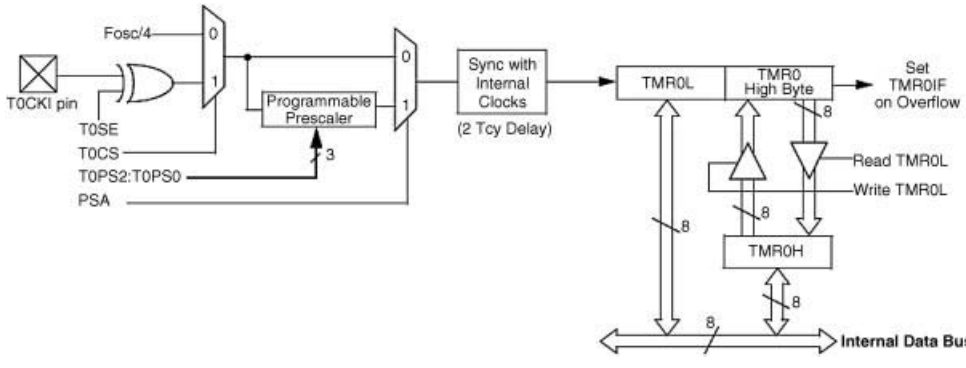


Figure 2.20 TIMER 0 block diagram in 16-bit operation. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T10SCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR10N
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer1 in one 16-bit operation
0 = Enables register read/write of Timer1 in two 8-bit operations

bit 6 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS1:T1CKPS0** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Figure 2.21 T1CON bit definitions. (Reproduced with permission from Microchip Inc)

When bit T1OSCEN is high, an internal oscillator is enabled. This oscillator is intended for 32 kHz real-time clock operation, although it can operate up to 200 kHz by connecting a crystal between pins TIOSI and TIOSO. A prescaler is available, as in TIMER 0, but with only 1, 2, 4 and 8 values.

Bit RD16 controls whether the timer is loaded in one 16-bit operation, or in two 8-bit operations. When RD16 is high, the timer is loaded, as in TIMER 0 16-bit mode. When RD16 is low, the timer is loaded as two 8-bit operations. Figure 2.22 shows the block diagram of TIMER 1.

2.2.7.3 TIMER 2

TIMER 2 is an 8-bit timer, controlled with SFR register T2CON. This timer has a 1: 16 prescaler, and also a 1: 16 postscaler. Figure 2.23 shows the bit definitions of T2CON.

An SFR register called PR2 is loaded with an 8-bit data, which is compared to timer register TMR2. If the two registers are equal, then an interrupt is generated. This way, the timer can be used to generate pulses at fixed time intervals. TIMER 2 is generally used for the PWM and CCP operations. Figure 2.24 shows the block diagram of TIMER 2.

www.it-ebooks.info

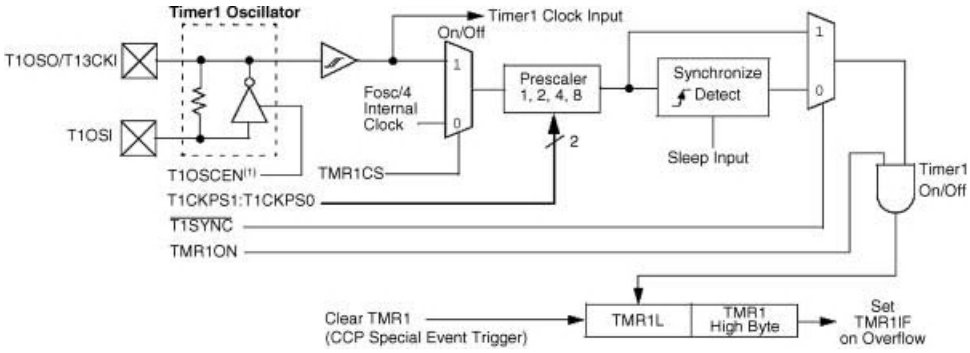


Figure 2.22 Block diagram of TIMER 1. (Reproduced with permission from Microchip Inc)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **T2OUTPS3:T2OUTPS0:** Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0000 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Figure 2.23 T2CON bit definitions. (Reproduced with permission from Microchip Inc)

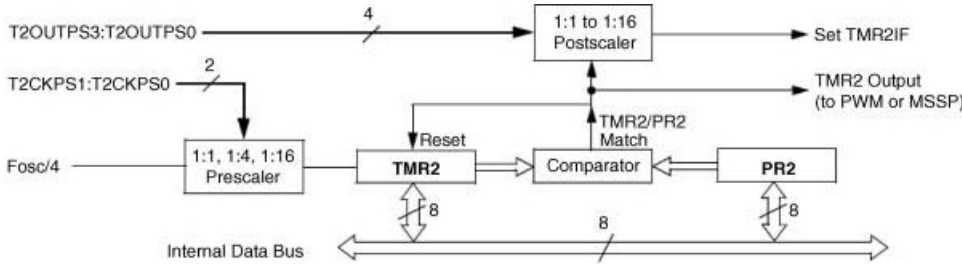


Figure 2.24 Block diagram of TIMER 2. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

2.2.7.4 TIMER 3

The operation of TIMER 3 is similar to TIMER 1, except that TIMER 3 can operate as a 16-bit timer, as a synchronous counter, and also as an asynchronous counter. The timer is controlled with SFR register T3CON, whose bit definition is shown in Figure 2.25.

2.2.8 Analogue-to-Digital Converter Module

The analogue to digital converter (A/D converter) module converts external analogue signals into digital form, so that they can be processed by the microcontroller. In general, an A/D converter can either be unipolar or bipolar. Unipolar A/D converters accept only positive input signals, while bipolar A/D converters can accept both negative and positive input signals. Bipolar A/D converters are frequenctly used in real-time DSP applications.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register read/write of Timer3 in one 16-bit operation
0 = Enables register read/write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the capture/compare clock source for the CCP madules
01 = Timer3 is the capture/compare clock source for CCP2;
Timer1 is the capture/compare clock source for CCP1
00 = Timer1 is the capture/compare clock source for the CCP madules

bit 5-4 **T3CKPS1:T3CKPS0:** Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the device clock comes from Timer1/Timer3.)
When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from Timer1 oscillator or T13CK (on the rising edge after the frist falling edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR3ON:** Timer3 On bit
1 = Enables Timer3
0 = Stops Timer3

Figure 2.25 T3CON bit definitions. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

The PIC18F2410 microcontroller contains 10 unipolar A/D converter channels, each having 10-bit resolution, that is 0 to 1023 steps. These A/D converters are usually operated from a +5 V reference voltage, giving a step size of

$$(5\,\mathrm{V}/1023) = 4.89\,\mathrm{mV} \tag{2.5}$$

Thus, the minimum voltage change that can be detected using such an A/D converter is 4.89 mV.

Figure 2.26 shows the block diagram of the PIC18F2410 A/D converter module. Although there are ten channels, there is only one actual A/D converter and the analogue inputs are multiplexed, only one channel being active at any given time.

SFR register ADCON0 is used to select a channel and start the actual conversion process. Figure 2.27 shows the bit definitions of register ADCON0. Bits CHS3: CHS0 select the analogue channel to be converted into digital. Bit ADON should be set to 1 to turn on the A/D converter module. Bit GO/DONE is set to 1 to start the conversion. During the

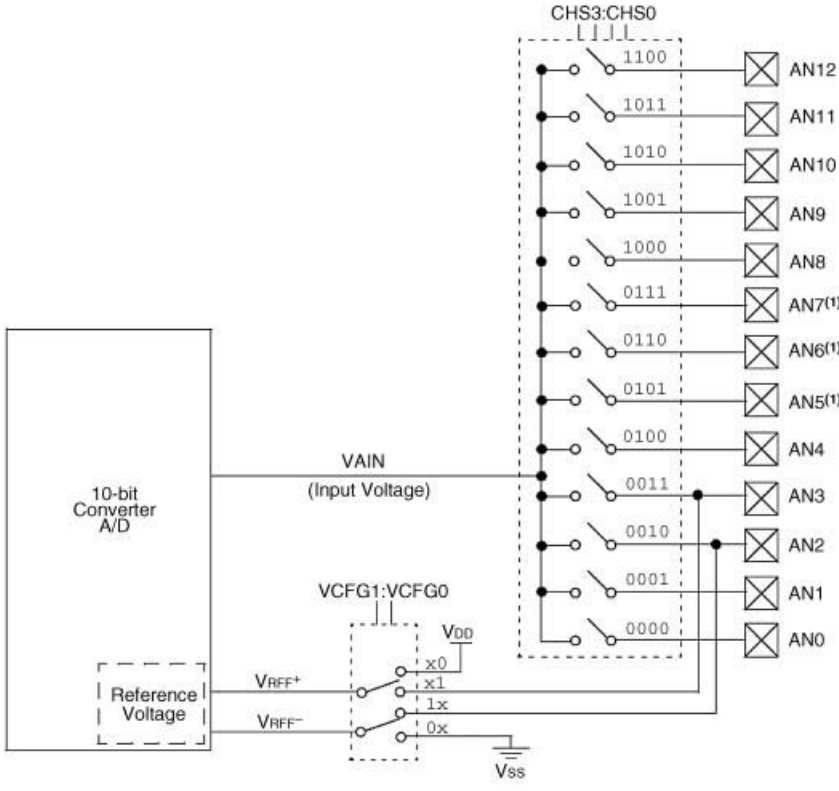


Figure 2.26 Block diagram of the PIC18F2410 A/D converter. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-2 **CHS3:CHS0:** Analog Channel Select bits

0000 = Channel 0 (AN0)
0001 = Channel 1 (AN1)
0010 = Channel 2 (AN2)
0011 = Channel 3 (AN3)
0100 = Channel 4 (AN4)
0101 = Channel 5 (AN5)(1,2)
0110 = Channel 6 (AN6)(1,2)
0111 = Channel 7 (AN7)(1,2)
1000 = Channel 8 (AN8)
1001 = Channel 9 (AN9)
1010 = Channel 10 (AN10)
1011 = Channel 11 (AN11)
1100 = Channel 12 (AN12)
1101 = Unimplemented(2)
1110 = Unimplemented(2)
1111 = Unimplemented(2)

Figure 2.27 ADCON0 bit definitions. (Reproduced with permission from Microchip Inc)

conversion process, this bit is cleared automatically and is set to 1 at the end of the conversion.

SFR register ADCON1 is used to configure the I/O ports, either as digital or as analogue. The bit definitions of ADCON1 are shown in Figure 2.28.

Bits VCFG1 and VCFG0 are used to select the negative and positive reference voltages. In most applications, VCFG1 is set to VSS (i.e. ground) and VCFG0 is set to VDD (i.e. +5 V reference voltage). Bits PCFG3:PCFG0 configure the analogue I/O ports. For example,

U-0	U-0	R/W-0	R/W-0	R/W-0(1)	R/W(1)	R/W(1)	R/W(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit (V_{REF-} source)
1 = V_{REF-} (AN2)
0 = V_{SS}

bit 4 **VCFG0:** Voltage Reference Configuration bit (V_{REF+} source)
1 = V_{REF+} (AN3)
0 = V_{DD}

bit 3-0 **PCFG3:PCFG0:** A/D port Configuration Control bits:

Figure 2.28 ADCON1 bit definitions. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Figure 2.29 PCFG3:PCFG0 bit definitions

clearing all these bits configure all the ten channels as analogue inputs. Figure 2.29 shows the PCFG3:PCFG0 bit definitions.

The A/D converter module has another SFR register called ADCON2, which is used to select the A/D result format and the A/D clock frequency. The bit definitions of ADCON2 are shown in Figure 2.30. The high and low bytes of the converted data are stored in registers ADRESH and ADRESL, respectively. Bit ADFM is an important bit. When set to 1, the A/D converter result is right justified, and when cleared, the result is left justified, as shown in Figure 2.31. In most applications, the result is right justified with zeroes filled in the upper 6 bits.

The A/D converter can be operated in two modes: polling, and interrupt driven. In polling mode, a conversion is started, and the end-of-conversion status is checked continually until the conversion is complete. In interrupt mode, an interrupt is generated as soon as the conversion is complete. The interrupt driven mode offers a much higher response time, but is generally more difficult to configure.

The steps in carrying out an A/D conversion are as follows:

- Set TRIS of the corresponding channel to 1 to configure the channel for input.
- Configure ADCON1 accordingly, to configure the required I/O ports as analogue.
- Configure ADCON2 bits to select the result format and the conversion clock.
- Configure ADCON0 to turn on the A/D converter module and select the required channel.
- Configure A/D converter end-of-conversion interrupt (if required).
- Set the GO/DONE bit of ADCON0 to start conversion.
- Wait until GO/DONE bit is set, or until an interrupt is generated (if enabled).

www.it-ebooks.info

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 **Unimplemented:** Read as '0'

bit 5-3 **ACQT2:ACQT0:** A/D Acquisition Time Select bits
111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

bit 2-0 **ADCS2:ADCS0:** A/D Conversion Clock Select bits
111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

Figure 2.30 ADCON2 bit definitions. (Reproduced with permission from Microchip Inc)

- Read the converted data into ADRESH and ADRESL.
- Repeat steps, as required for a new conversion.

It is important to take extra care when converting very fast varying analogue signals into digital. A sample-and-hold amplifier may be needed to hold the signal from changing before the conversion is started.

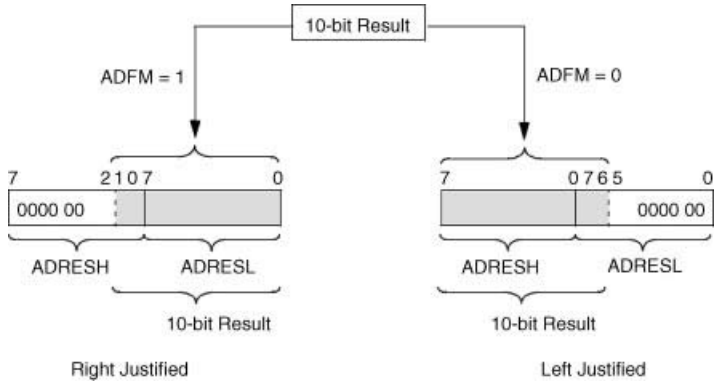


Figure 2.31 A/D converter result formatting

www.it-ebooks.info

2.2.9 *Special Features of the CPU*

The PIC18F2410 microcontroller includes several features intended to maximise reliability and minimise costs by eliminating the use of external components. Some of the important features are the configuration registers and the watchdog timer.

The configuration registers are usually configured during physical programming of the microconroller chip. These registers hold various important microcontroller features, such as the oscillator modes, brown-out detector configuration, watchdog configuration, and so on. In this section we look at the two most important configuration registers, CONFIG1H and CONFIG2H. Details of other configuration registers can be found in manufacturers' data sheets.

CONFIG1H is used to select the microcontroller clock mode and the clock switching logic. The bit definitions of this register are shown in Figure 2.32.

Configuration register CONFIG2H is used to configure the watchdog timer module. Figure 2.33 shows the bit definitions of this register. The watchdog is enabled by setting bit WDTEN to 1. Bits WDTPS3:WDTPS0 select the watchdog timer postscaler bits. The range can be selected from 1 to 32 768. The watchdog is clocked by the INTRC clock source. The nominal watchdog period is 4 ms. This value is multiplied by selecting a 16-bit postscaler value. Thus, the timing ranges from 4 ms to $4 \times 32\,768$ ms, or 131.072 s (2.18 min). If the WDTEN bit is cleared, then the watchdog timer can be enabled in software by setting bit SWDTEN of SFR register WDTCON. A soft reset action is performed if the watchdog

R/P-0	R/P-0	U-0	U-0	R/P-0	R/P-1	R/P-1	R/P-1
IESO	FCMEN	—	—	FOSC3	FOSC2	FOSC1	FOSC0
bit 7							bit 0

bit 7 **IESO:** Internal/External Oscillator Switchover bit
1 = Oscillator Switchover mode enabled
0 = Oscillator Switchover mode disabled

bit 6 **FCMEN:** Fail-Safe Clock Monitor Enable bit
1 = Fail-Safe Clock Monitor enabled
0 = Fail-Safe Clock Monitor disabled

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **FOSC3:FOSC0:** Oscillator Selection bits
11xx = External RC oscillator, CLKO function on RA6
101x = External RC oscillator, CLKO function on RA6
1001 = Internal oscillator block, CLKO function on RA6, port function on RA7
1000 = Internal oscillator block, port function on RA6 and RA7
0111 = External RC oscillator, port function on RA6
0110 = HS oscillator, PLL enabled (Clock Frequency = 4 x FOSC1)
0101 = EC oscillator, port function on RA6
0100 = EC oscillator, CLKO function on RA6
0011 = External RC oscillator, CLKO function on RA6
0010 = HS oscillator
0001 = XT oscillator
0000 = LP oscillator

Figure 2.32 CONFIG1H bit definitions. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4-1 **WDTPS3:WDTPS3:** Watchdog Timer Postscale Select bits

1111 = 1:32,768
1110 = 1:16,384
1101 = 1:8,192
1100 = 1:4,096
1011 = 1:2,048
1010 = 1:1,024
1001 = 1:512
1000 = 1:256
0111 = 1:128
0110 = 1:64
0101 = 1:32
0100 = 1:16
0011 = 1:8
0010 = 1:4
0001 = 1:2
0000 = 1:1

bit 0 **WDTEN:** Watchdog Timer Enable bit

1 = WDT enabled
0 = WDT disabled (control is placed on the SWDTEN bit)

Figure 2.33 CONFIG2H bit definitions. (Reproduced with permission from Microchip Inc)

times out. The watchdog module is usually used in real-time and time-critical applications to make sure that all time-critical applications complete their tasks in the specified times.

Figure 2.34 shows the block diagram of the watchdog timer module.

2.2.10 *Interrupts*

An interrupt is an external or internal event that requires the CPU to stop its normal execution and run a program related to the interrupting event. Internal interrupts are usually generated when a timer overflows, when the A/D conversion is complete, when a character is

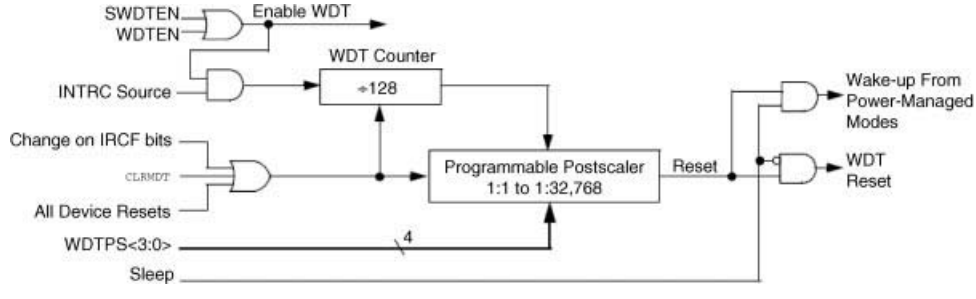


Figure 2.34 Block diagram of the Watchdog timer module. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

received by the USART, and so on. External interrupts are usually generated when certain pins of the microcontroller change state.

Interrupts can be useful in many applications, such as:

Servicing time critical applications. Real-time applications need immediate attention of the CPU. For example, in signal processing applications, the CPU has to receive signals from external sources whenever a signal is available. Also, for safety reasons, it may be required to shut down a plant whenever there is power failure or fire. In such applications, the CPU is required to stop whatever it is doing and service the interrupting device.

Performing scheduled tasks. There are many applications that require the CPU to perform scheduled tasks, such as updating the real-time clock. These tasks are important and must be serviced at the exact times.

Task switching. In many multi-tasking applications, it is required to service each task for a certain amount of time. This is usually achieved using timer interrupts where each time an interrupt occurs the running task is saved and a new task is started.

Preventing CPU from being tied up. In some applications, the CPU is required to perform continuous checks of I/O devices. While performing these checks, the CPU cannot perform other duties. By moving the checking into interrupt routines, the checking can be done in the background and the CPU is free to carry out other tasks. For example, a multi-digit 7-segment display needs to be refreshed continuously. If this task is done in the main program, then the CPU cannot do other tasks. By moving the refreshing operation into the timer interrupt routine, the CPU is free to do other tasks.

The PIC18F2410 microcontroller has multiple interrupt sources, such as external interrupts via port pins RB0 (INT0), RB1 (INT1), and RB2 (INT2), PORT B interrupt when any of the pins RB4 to RB7 change state, Timer interrupts, A/D converter interrupts, USART interrupts, and so on. The interrupt sources are divided into *core* interrupt sources and *peripheral* interrupt sources. The core interrupt sources are the external interrupts and the TMR0 interrupts. Peripheral interrupt sources are the other external and internal interrupt sources. More details can be obtained from the manufacturer's data sheets. In this book we briefly look at the interrupt mechanisms.

There are 10 registers that control the interrupt operations:

- RCON.
- INTCON, INTCON2, INTCON3.
- PIR1, PIR2.
- PIE1, PIE2.
- IPR1, IPR2.

The interrupts can be divided into two categories: high-priority group and low-priority group. If a certain device requires closer attention, then it should be set as a high-priority device. If the priority is not important, then it is advisable to set all interrupts as low-priority.

The interupt priority feature is enabled by setting the IPEN bit of register RCON. If IPEN = 0, then the interrupt feature is not enabled and the processor behaves as if all interrupts are at the same priority group (this is the case with the PIC16 microcontroller family).

When the interrupt priority is enabled, two bits of register INTCON are used to enable interrupts globally. The GIEH bit enables all interrupts whose priority bits are set. Setting

www.it-ebooks.info

GIEL enables all interrupts whose priority bits are cleared. High-priority interrupts vector to address 0×0018, while the low-priority interrupts vector to address 0×0008. Interrupt sources have three bits to control their operation:

- A flag bit to indicate whether an interrupt has occured. This bit has a name ending in . . . **IF,** e.g. TMR0IF;
- Enable bit to enable or disable an interrupt source. This bit has a name ending in . . . **IE**, e.g. TMR0IE;
- A priority bit to select the interrupt priority. This bit has a name ending in . . . **IP**, e.g. TMR0IP.

Figure 2.35 shows the bit definitions of register INTCON, which is the main interrupt control register. The core interrupt sources are controlled by registers INTCON, INTCON2 and INTCON3. The bit definitions of INTCON2 and INTCON3 registers are given in Figure 2.36 and Figure 2.37 respectively.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts
When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts
When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interruput

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

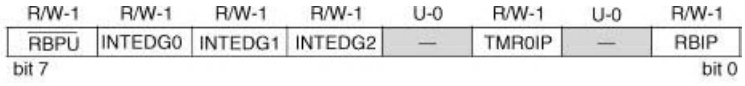
bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Figure 2.35 INTCON bit definitions. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info



R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0:** External Interrupt 0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1:** External Interrupt 1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2:** External Interrupt 2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

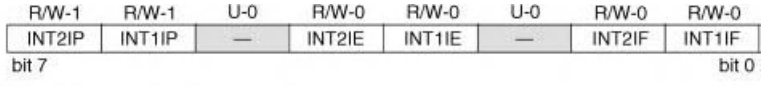
bit 3 **Unimplemented:** Read as '0'

bit 2 **TMR0IP:** TMR0 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **Unimplemented:** Read as '0'

bit 0 **RBIP:** RB Port Change Interrupt Priority bit
1 = High priority
0 = Low priority

Figure 2.36 INTCON2 bit definitions. (Reproduced with permission from Microchip Inc)



R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Figure 2.37 INTCON3 bit definitions. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

2.2.10.1 Interrupts with Priority Disabled

When the interrupt priority feature is disabled, the following conditions must be satisfied for an interrupt to be accepted by the processor:

- Disable priority feature, IPEN = 0.
- Set the GIE bit of INTCON.
- Clear the interrupt flag of the interrupt source.
- Enable the interrupt bit of the interrupting device. If the interrupt is from a core device, for example the TMR0, then TMR0IE bit of INTCON must be set to 1. If the interrupt is from a peripheral device, for example the A/D converter, then enable the PEIE bit of INTCON as well as the device interrupt enable bit ADIE of the appropriate PIE register.

2.2.10.2 Interrupts with Priority Enabled

- Enable priority feature, IPEN = 1.
- Set bits GIEH and GIEL of INTCON.
- Clear the interrupt flag of the interrupt source.
- Set the priority level using the corresponding IPR register.
- Set the interrupt enable bit, using either INTCON, INTCON2, INTCON3 or PIEI registers.

As an example, the steps required to set TMR0 as a high-priority interrupt are:

- Enable the priority feature. Set IPEN = 1.
- Enable TMR0 interrupts. Set TMR0IE = 1.
- Enable TMR0 high priority. Set TMR0IP = 1.
- Clear TMR0 interrupt flag. Set TMR0IF = 0.
- Enable global interrupts. Set GIEH = 1.

2.2.11 Pulse Width Modulator Module

The pulse width modulator module (PWM) generates a PWM output waveform with 10-bit resolution from certain pins of PIC microcontrollers. Some microcontrollers have only one PWM module, while some others can have three or more. The PWM pin is identified by the letters CCPx, where x is the number of the module (e.g. CCP1).

A PWM is basically a square wave signal with a specified period and duty cycle (see Figure 2.38).

On the PIC18F2410 microcontroller there is only one PWM module, with its pin CCP1 shared with I/O port pin RC2. On this microcontroller, the PWM module timing is controlled by Timer 2.

A PWM waveform has two parameters: the period and the duty cycle. Both of these parameters must be programmed before the required waveform can be generated.

The PWM period is given by

$$PWM\ period = (PR2 + 1) * TMR2PS * 4 * T_{OSC} \tag{2.6}$$

www.it-ebooks.info

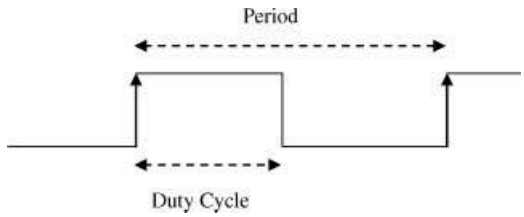


Figure 2.38 Typical PWM waveform

or

$$PR2 = \frac{PWM\ period}{TMR2PS * 4 * T_{OSC}} - 1 \tag{2.7}$$

where

PR2 is the value loaded into Timer 2 register.
TMR2PS is the Timer 2 prescaler value (2, 4 or 16).
T_{OSC} is the clock oscillator period (seconds).

The duty cycle consists of 10 bits. The 8 most significant bits are loaded into the CCPR1L register and the 2 least significant bits are loaded into bits 4 and 5 of the CCP1CON register. The duty cycle (in seconds) is given by

$$PWM\ duty\ cycle = (CCPR1L : CCP1CON < 5 : 4 >) * TMR2PS * T_{OSC} \tag{2.8}$$

or

$$CCPR1L : CCP1CON < 5 : 4 >= \frac{PWM\ duty\ cycle}{TMR2PS * T_{OSC}} \tag{2.9}$$

The steps to configure the PWM are then:

- Specify the required period and duty cycle.
- Choose a value for Timer 2 prescaler (TMR2PS).
- Calculate the value to be written into PR2 register using the formula given.
- Calculate the value to be loaded into CCPR1L and CCP1CON registers using the formula given.
- Clear bit 2 of TRISC to make CCP1 pin an output pin.
- Configure the CCP1 module for PWM operation using register CCP1CON.

Bits 0–3 of CCP1CON register must be set to '1100' to enable the PWM module. Bits 4 and 5 of this register must be loaded with the two LSB bits of the duty cycle value.

An example is given below to show how the PWM module can be set up. In this example it is assumed that we are using a PIC18F2410 type microcontroller operated with a 4 MHz clock. We further assume that the required PWM period is 60 μs and the required duty cycle (ON time) is 30 μs. The steps required to configure the various registers for this operation are:

www.it-ebooks.info

- Using a 4 MHz clock, $T_{OSC} = 1/4 = 0.25 \times 10^{-6}$ μs.
- Assuming a Timer 2 prescaler factor of 4, we have:

$$PR2 = \frac{PWM\ period}{TMR2PS * 4 * T_{OSC}} - 1 \tag{2.10}$$

or

$$PR2 = \frac{60 \times 10^{-6}}{4 * 4 * 0.25 \times 10^{-6}} - 1 = 14 \text{ or } 0 \times 0E \tag{2.11}$$

Also

$$CCPR1L : CCP1CON < 5 : 4 >= \frac{PWM\ duty\ cycle}{TMR2PS * T_{OSC}} \tag{2.12}$$

or

$$CCPR1L : CCP1CON < 5 : 4 >= \frac{30 \times 10^{-6}}{4 * 0.25 \times 10^{-6}} = 30 \tag{2.13}$$

The equivalent of number 30 in 10-bit binary is '0000011110' or '00000111 10'

- Therefore, the value to be loaded into bits 4 and 5 of CCP1CON is '10'. Bits 2 and 3 of CCP1CON must be set to high for PWM operation. Therefore, CCP1CON must be set to bit pattern ('X' is 'don't care'):

XX001100 that is hexadecimal $0 \times C0$

- The value to be loaded into CCPR1L is '00000111', that is hexadecimal number 0×07. The required steps are summarised below:
- Load Timer 2 with prescaler of 4, that is load T2CON with 00000101, which is 0×05;
- Load $0 \times 0E$ into PR2;
- Load 0×07 into CCPR1L;
- Load 0 into TRISC (make CCP1 pin output);
- Load $0 \times C0$ into CCP1CON.

One period of the generated PWM waveform is shown in Figure 2.39.

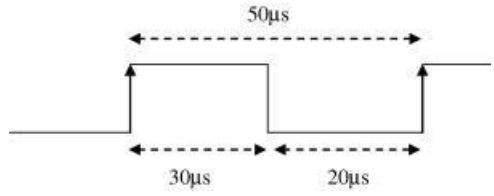


Figure 2.39 Generated PWM waveform

www.it-ebooks.info

2.3 Summary

This chapter has described the important features of the architecture of the PIC18F2410 microcontroller. The architecture of most other PIC18F families of microcontrollers are similar, having additional memories, additional I/O ports, or special functions such as CAN bus modules, USB modules, and so on.

The important parts of the PIC18F2410 microcontroller have been described, including the data memory map, program memory map, I/O ports, clock sources, timers, A/D converter, configuration registers, the watchdog, interrupts, and the PWM module.

Exercises

2.1 Describe the program memory map of the PIC18F2410 microcontroller. How much is the maximum addressable memory?

2.2 Describe the data memory map of the PIC18F2410 microcontroller. What is a memory bank? How many memory banks are there in the PIC18F2410 microcontroller?

2.3 What is an SFR register? Give an example.

2.4 Describe how a PIC18F2410 microcontroller can be operated from an external 8 MHz crystal. Draw the circuit diagram.

2.5 Explain how the PLL can be used to increase the clock frequency in a PIC18F2410 microcontroller.

2.6 Show which bits should be set to operate a PIC18F2410 microcontroller from the internal 4 MHz clock.

2.7 Draw a circuit diagram to show how a PIC18F2410 microcontroller can be reset using an external reset button.

2.8 Draw the block diagram of a typical I/O port (without the peripheral functions) and explain how the input-output operations take place.

2.9 Explain the importance of reading a port value using the LAT register instead of the standard PORT register.

2.10 In a non-time-critical application, it is required to operate a PIC18F2410 microcontroller using an external resistor and a capacitor for timing. Assuming the required clock frequency is 5 MHz, what will be the values of the resistor and the capacitor? Draw the circuit diagram to show how these timing components can be connected to the microcontroller.

2.11 In an application it is required to generate a delay of 250 μs using TIMER 0. Calculate the value to be loaded into register TMR0L, assuming that the microcontroller is operated from a 6 MHz clock, and TIMER 0 is operated in 8-bit mode.

2.12 In a TIMER 0 based application, register TMR0L is loaded with 250. Assuming the microcontroller clock rate is 8 MHz, and the prescaler is 64, calculate the time it will take to overflow the timer.

2.13 In an application it is required to use the watchdog to reset the microcontroller every 250 μs. Explain how this can be achieved.

2.14 Explain why the watchdog is important in time-critical applications.

2.15 Explain what a configuration register is. How can the configuration register be loaded?

www.it-ebooks.info

2.16 Draw the circuit diagram to show how an external clock can be connected to a PIC18F2410 microcontroller to provide clock pulses.
2.17 In an application it is required to operate a PIC18F2410 microcontroller with a clock of 4 MHz. Describe all different ways that this can be achieved and draw the circuit diagram for each case.
2.18 In a time-critical application it is required to operate a PIC18F2410 microcontroller with a 40 MHz clock. Describe how this can be achieved using the PLL and draw the circuit diagram.
2.19 Assuming that an 8 MHz clock is used with a prescaler of 256, what is the maximum time for the timer to overflow assuming operation of the timer in 16-bit mode?
2.20 It is required to set PORT A of a PIC18F2410 microcontroller, such that the pins at the lower nibble are inputs and those at the upper nibble are outputs. What value must be loaded into the TRISA register?
2.21 In a typical A/D converter application, the reference voltage is +5 V. If the digital value of the input signal read is hexadecimal $0 \times 3F$, what is the voltage in m V?
2.22 In an A/D application, the reference voltage is +5 V. If a 2 V signal is applied to one of the analogue inputs, what is the digital value of the signal read?
2.23 Explain why interrupts are important in microcontroller systems.
2.24 How many different types of interrupt sources are there in the PIC18 series of microcontrollers.
2.25 Explain the differences between high-priority and low-priority interrupts.
2.26 It is required to set up the timer TMR0 to generate interrupts on overflow. Explain which registers need to be set up
2.27 Explain the steps required to generate a PWM signal.
2.28 It is required to generate a PWM signal with a period of 100 μs and duty cycle of 80 μs. Assuming a microcontroller clock frequency of 4 MHz, calculate the values to be loaded into various registers.

www.it-ebooks.info

3

C Programming Language

Microcontrollers can be programmed using one of several high-level languages. Some of the commonly used high-level languages are BASIC, PASCAL and C. Because of its ease of use and its power, C is probably the most commonly used high-level language for microcontroller programming. C has been used for the past 10 to 15 years and has gone through tremendous growth. Today, there seems to be an endless supply of low-cost, high-quality C compilers directed to any type of computer, from the PC to the smallest 8-pin microcontroller chip.

Before the development of the C language, microprocessors and microcontrollers were being programmed using the native Assembly language of the target chip. Assembly language was a difficult language to learn. Also, it was difficult and time consuming to develop and maintain complex applications using the Assembly language. For example, developing a mathematical application using floating point arithmetic took days, if not weeks. The developed code was so large that it was difficult to modify or maintain it. In addition, the code was specific for the target processor. If it was required to upgrade to a different processor then, in most cases, it was necessary to re-write the complete code, consuming a lot of time and effort. C, on the other hand, is a portable language. In general, a C program written for a specific processor can easily be modified and used for another processor. Highly complex programs can be developed and maintained using the C language.

The main objective of this chapter is to introduce new readers to the C language, through a simple overview of its fundamental features, in an attempt to start them programming early. The C language is described in a tutorial way, with many examples, in such a way that the readers can gain confidence and start coding immediately. However it is important for the reader to appreciate that C is a rich language with many features, and this is not a text book on C language, so only the parts relevant to future chapters of the book will be covered here.

3.1 C Languages for Microcontrollers

There are basically three types of C compilers in the market for the PIC microcontrollers, depending upon the type of microcontroller used. Some compilers are available for the low-end microcontrollers, such as the PIC10/12/16 series. Some compilers are available for the

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

mid-range microcontrollers, such as the PIC18 series. Some compilers are specifically designed for the mid-range PIC24 series of microcontrollers, and a few compilers are available for the high-end PIC32 series of microcontrollers.

In this book we shall be using the mid-range PIC18 microcontrollers in our designs and examples. There are several C compilers in the market developed for the PIC18 series of microcontrollers. Some of the popular C compilers used in the development of educational, domestic, commercial or industrial PIC18 microcontroller based projects are:

- mikroC Pro for PIC C compiler;
- PICC18 C compiler;
- C18 C compiler;
- CCS C compiler.

mikroC Pro for PIC C compiler has gained much popularity in recent years, because of its low cost, ease of use and full support for various development boards. This compiler has been developed by *MikroElektronika* (Web site: www.mikroe.com) and is one of the easy-to-learn compilers with rich resources, such as a large number of built-in library functions. The compiler provides an integrated development environment (IDE) with built-in editor, compiler, simulator and an in-circuit-debugger (e.g. mikroICD). Users can write a program using the built-in editor, then compile and simulate the program with the click of a few buttons. The final working program can be downloaded to the target microcontroller by clicking a button. In addition to the simulator, users can carry out real-time debugging of their applications using the built-in in-circuit-debugger. A demo version of the compiler with a 2 KB program limit is available from MikroElektronika for educational use or for learning the basic features of the compiler. Users can upgrade to the full version after making the necessary payment. The full version requires a dongle to be connected to the USB port for compiling programs greater than 2 KB, or users can choose to tie the compiler to their PCs by registering using the serial number of their PCs. In addition, the company offers low-cost integrated packages, including the compiler and a hardware development board. In this book we shall mainly concentrate on the use of the microC compiler, and all of the projects are based on this compiler.

PICC18 C compiler is another popular C compiler, developed by *Hi-Tech Software* (Web site: www.htsoft.com). This compiler has two versions: the standard compiler and the professional version. A powerful simulator and an ITD (Hi-Tide) are provided by the company. PICC18 is supported by the PROTEUS simulator (www.labcenter.co.uk), which can be used to simulate PIC microcontroller based systems with various peripheral devices, such as LEDs, motors, buttons, and so on. A limited period demo version of this compiler is available from the developers' Web site.

C18 C compiler is a product of the *Microchip Inc.* (Web site: www.microchip.com). A limited period demo version, and a limited functionality version with no time limit of C18, are available from the Microchip Web site. C18 includes a simulator, and supports hardware and software development tools, such as in-circuit-emulators (e.g. ICE2000) and in-circuit-debuggers (e.g. ICD2, ICD3, Real-Ice, and so on). C18 includes a large number of library functions that can be used during program development. The compiler is based on an integrated environment where users can create programs, compile them, and then download to the target microcontroller using either a suitable programmer (or debugger) or using a development board with built-in PIC microcontroller programmer hardware.

www.it-ebooks.info

CCS C compiler has been developed by the *Custom Computer Systems Inc* (Web site: www.ccsinfo.com). The company provides a limited period demo version of their compiler for users who may want to evaluate the compiler. CCS compiler provides a large number of built-in functions and supports an in-circuit-debugger (e.g. ICD-U40), which aids greatly in the development of PIC18 microcontroller based systems. CCS C compiler is very fast and produces compact code. The compiler is fully compatible with various hardware development boards offered by the developers. In addition, the company offers low-cost integrated packages, including the compiler and a hardware development board. The syntax of the compiler is somewhat different to the other C compilers. A large number of examples and tutorials are provided by the developers to help users in their projects.

In this book we shall be looking at the features of the popular and powerful mikroC Pro for PIC programming language, and this language will be used in all of the projects in the book.

3.2 Your First mikroC Pro for PIC Program

Figure 3.1 shows a very simple mikroC Pro for PIC program. This program turns ON all the 8 LEDs connected to port B of a PIC microcontroller. Then, after a 500 millisecond delay, all the 8 LEDs are turned OFF. Do not worry if you do not understand the operation of this program at this stage, as all will be clear as we progress through this chapter. Some of the elements used in Figure 3.1 are described in detail here.

3.2.1 Comments

Comments are used in programs to clarify the operation of the program. Although the use of comments are optional, it is strongly recommended that you use as many comments as possible in your programs, as comments make your programs readable and easily maintainable. Imagine how hard it would be to write a complex program with no comments and then

```
/*=========================================================================

                              TURN ON-OFF ALL LEDS
                              --------------------------------------

This program turns ON all the 8 LEDs connected to PORT B of a PIC microcontroller. Then,
After 500 millisecond delay all 8 LEDs are turned OFF

Author:         D. Ibrahim
File:           LED.C
Date:           October, 2011
Modifications:

=========================================================================*/

void main()
{
        TRISB = 0;                      // Configure PORT B as output
        PORTB = 0xFF;                   // Turn ON all 8 LEDs
        Delay_Ms(500);                  // Wait 500 ms
        PORTB = 0;                      // Turn OFF all 8 LEDs
}
```

Figure 3.1 A very simple mikroC Pro for PIC program

www.it-ebooks.info

try to modify the program after several months. All the comment lines are ignored by the compiler.

In mikroC Pro for PIC language, comments can be of two types: long comments and short comments. Long comments start with the character pair:

```
/*
```

and end with the character pair:

```
*/
```

Long comments are commonly used at the beginning of a program to describe the program details, such as what the program does, what type of hardware is used, who the author is, the date program was created, filename of the program, version history, and so on. You can see the use of long comments at the beginning of our simple program. Long comments are also used inside a program to describe the operation of part of the program, for example the parameters of functions, the algorithm used, and so on.

Short comments start with the character pair:

```
//
```

Short comments are not terminated with a character and they can be used in a single line, starting anywhere in the line. These comments are generally used after program statements and they describe what the statement does. Examples of short comments can be seen in our simple program in Figure 3.1.

3.2.2 Beginning and Ending a Program

In mikroC Pro for PIC language, a program starts with the keywords:

```
void main()
```

After this, a curly opening bracket is used to indicate the start of program body. The program is terminated with a curly closing bracket. Thus, the structure of a program is (see Figure 3.1):

```
void main()
{
        Program body
}
```

The program body consists of program statements. Each program statement must be terminated with a semicolon (';') character to indicate the end of the statement, otherwise an error will be generated by the compiler:

```
k = 50;     //correct
i = k+5;    //correct
k = 50      //error
```

www.it-ebooks.info

3.2.3 White Spaces

White spaces in programs consist of spaces, tabs, newline characters, and blanks. These characters are ignored by the compiler. Thus, the following lines are identical:

```
k = 20; p = 50;
```

or

```
k = 20;
p = 50;
```

or

```
k = 20;
     p =50;
```

or

```
k=20;
p=
   20;
```

In some applications, we may have a long string that we may want to extend over several lines. The backspace character (‘\’) is used to join strings that extend to several lines. For example:

```
"My new mikroC\
 Compiler"
```

parses into string ‘My new mikroC Compiler’.

3.2.4 Variable Names

In mikroC Pro for PIC language, variable names can begin with a letter or the underscore character (‘_’). Variable names can include any character a to z, A to Z, or 0 to 9. A variable name can be up to 31 characters long. Some examples of valid variable names are:

```
Total   Sum   Average   My_Variable   username   MaxTotal   _Name
```

and here are some invalid variable names:

```
%name       ?Total       7Sum       (Max       12count
```

www.it-ebooks.info

Table 3.1 mikroC Pro for PIC reserved names

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

The names are case sensitive and thus variables with lowercase names are different to variables with uppercase names. Thus, the following variables are all different:

```
Total   total   ToTal   TotaL   TOTAL    TOTal
```

3.2.5 *Reserved Names*

Some names in mikroC Pro for PIC are reserved for the compiler and these names can not be used as variable names. Table 3.1 gives a list of these reserved names. For example, the following variable names are illegal:

```
for   while   char   int   return   signed      const
```

3.2.6 *Variable Types*

mikroC Pro for PIC is a strictly typed language, which means that every object, expression or function must have a defined type before the program is compiled. mikroC Pro for PIC supports many pre-defined and user-defined data types, including signed and unsigned bytes and integers in various sizes, floating point numbers in various precisions, arrays, structures, and so on. The type of a variable defines how much memory space should be allocated for a variable in memory and how the bits of the variable should be manipulated. Types can be divided into two groups: Fundamental types and Derived types. The fundamental types represent types that cannot be split up into smaller parts. These types are *void*, *char*, *int*, *float* and *double*, together with *short*, *long*, *signed* and *unsigned variants*. The derived types are also known as structured types and they include *pointers*, *structures*, *arrays* and *unions*.

mikroC Pro for PIC language supports the fundamental variable types shown in Table 3.2. Examples of these variable types are given in this section.

unsigned char or **unsigned short int** are unsigned 8-bit variables, occupying only 1 byte in memory and having values in the range 0 to 255. In the following example, variable *Sum* is assigned value 225:

```
unsigned char Sum = 225;
```

www.it-ebooks.info

Table 3.2 mikroC Pro for PIC variable types

Type	Size (bits)	Range
unsigned char	8	0 to 255
unsigned short int	8	0 to 255
unsigned int	16	0 to 65 535
unsigned long int	32	0 to 4 294 967 295
signed char	8	−128 to 127
signed short int	8	−128 to 127
signed int	16	−32 768 to 32 767
signed long int	32	−2 147 483 648 to 2 147 483 647
float	32	±1.17549435082E-38 to ±6.80564774407E38
double	32	±1.17549435082E-38 to ±6.80564774407E38
long double	32	±1.17549435082E-38 to ±6.80564774407E38

or

```
unsigned char Sum;
Sum = 225;
```

unsigned int variables are unsigned 16-bit variables, occupying 2 bytes in memory and having values in the range 0 to 65 535. In the following example, variable *Total* is assigned value 64 500:

```
unsigned int Total = 64500;
```

unsigned long int variables are unsigned 32-bit variables, occupying 4 bytes in memory and having values in the range 0 to 4 294 967 295. In the following example, variable *Sum* is assigned value 4 200 000 000:

```
unsigned long int Sum = 4200000000;
```

signed char or **signed short int** variables are signed 8-bit variables, occupying only 1 byte in memory and having values in the range −128 to +127. In the following example, variable *Total* is assigned value −240:

```
signed char Total = -240;
```

signed int variables are 16-bit variables, occupying 2 bytes in memory and having values in the range −32 768 to +32 767. In the following example, variable *Sum* is assigned value −31 500:

```
signed int Sum = -31500;
```

www.it-ebooks.info

signed long int variables are 32-bit variables, occupying 4 bytes in memory and having values in the range −2 147 483 648 to +2 147 483 647. In the following example, variable *Sum* is assigned value 2 050 480 000:

```
signed long int Sum = 2050480000;
```

Floating point number data types are **float**, **double** and **long double**. mikroC Pro for PIC implements the floating point numbers using the Microchip AN575 32-bit format, which is IEEE 754 compliant. The floating point numbers have values in the range ±1.17549435082E-38 to ±6.80564774407E-38. In the following example, variable *Volume* is assigned value 23.45:

```
float Volume = 23.45;
```

or

```
float Volume;
Volume = 23.45;
```

3.2.7 Constants

Constants are very important in mikroC Pro for PIC programs, especially if the RAM data memory has limited size. Constant variables are stored in the microcontroller flash program memory, thus freeing valuable RAM memory space. In mikroC Pro for PIC, constants can be characters, integers, floating point numbers, strings and enumerated variables.

3.2.7.1 Character Constants

A character constant occupies a single byte in the program memory. The constant is declared by specifying the character within a single quote mark. In the following example, variable *FirstName* is declared as a constant character and is assigned the value 'D':

```
const FirstName;
FirstName = 'D';
```

3.2.7.2 Integer Constants

Integer constants occupy 2 bytes in memory. These constants can be specified using decimal, hexadecimal, octal or binary bases. The data type of a constant is derived by the compiler automatically, depending upon the value of the constant. For example, a constant with a value 130 is stored as an *unsigned char*, a constant with a value 12 000 is stored as an *unsigned int*, and a constant with a value −22 500 is stored as a *signed int*.

In the following example, *MIN* and *MAX* are defined as constants 0 and 200, respectively:

```
const MIN = 0;
const MAX = 200;
```

www.it-ebooks.info

Hexadecimal numbers have the range 0 to 9 and A to F. Hexadecimal constants are specified by inserting characters '0x' or '0X' in front of the number. In the following example, constant *MAX* is defined to have the hexadecimal value FFF:

```
const MAX = 0xFFF;
```

Octal numbers have the range 0 to 7. Octal constants are specified by inserting number '0' in front of the number. In the following example, constant *MAX* is defined to have the octal value 177:

```
const MAX = 0177;
```

Binary numbers can be 0 or 1. These numbers are specified by inserting characters '0b' or '0B' in front of the number. In the following example, constant *MIN* is defined to have the binary value 01 101 111:

```
const MIN = 0b01101111;
```

3.2.7.3 Floating Point Constants

Floating point constants are non-integer constants having a decimal part, a dot and the fractional part. In addition, for very large or very small numbers, the exponent part can be specified by inserting characters 'e' or 'E' with the value of the exponent at the end of the number. In the following example, variable **MIN** is given the value 0.15E-2, and **MAX** is given the value 25.5E10:

```
const MIN = 0.15E-2;
const MAX = 25.5E10;
```

3.2.7.4 String Constants

String constants consist of collections of characters enclosed within double quotes. An example string constant is:

```
"This is a string"
```

As we shall see in later sections, strings are made up of character arrays.

3.2.7.5 Enumerated Constants

An enumeration data type is used for representing an abstract, discreet set of values with appropriate symbolic names. Enumeration makes a program easier to follow.

Variables of the *enum* type are declared the same as other variables. An example enumeration declaration is given below:

```
enum colours
{
```

www.it-ebooks.info

```
        Black,
        Red,
        Green,
        Blue,
        Cyan
    } clr;
```

In the above example, Black = 0, Red = 1, Green = 2, and so on. Identifier *clr* can take any value of the specified colours, or any integer value:

```
    clr = Red;        //clr = 1
```

or

```
    clr = 1;          //same as above
```

The order of constants in an *enum* type can be explicitly re-arranged using specific values. Any names without initialisers will be increased by 1 with respect with the previous value. An example is given below:

```
    enum colours
    {
        Black,          //value 0
        Red,            //value 1
        Green = 4,      //value 4
        Blue,           //value 5
        Cyan            //value 6
    } clr;
```

Another example is given below:

```
    enum Weekdays
    {
         MON = 1;       //value 1
         TUE,           //value 2
         WED,           //value 3
         THU,           //value 4
         FRI,           //value 5
         SAT,           //value 6
         SUN            //value 7
    }
```

3.2.8 *Escape Sequences*

Some of the control characters in the ASCII table are non-printable and are known as the escape sequences. For example, the character '\n' represents the new-line character, which causes the cursor to jump to the next line. Table 3.3 gives a list of the commonly used escape

www.it-ebooks.info

Table 3.3 Commonly used escape sequences

Escape Sequence	Hex Value	Character
\a	0×07	BEL (bell)
\b	0×08	BS (backspace)
\t	0×09	HT (horozontal tab)
\n	0×0A	LF (linefeed)
\v	0×0B	VT (vertical feed)
\f	0×0C	FF (formfeed)
\r	0×0D	CR (carriage return)
\xH		String of hex digits

sequences. Notice that the escape sequence characters can also be obtained by specifying their values. For example, the hexadecimal '0×0A' can be used to specify the new line.

3.2.9 Volatile Variables

Volatile declaration is particularly important in interrupt based applications. The qualifier volatile implies that a variable may change its value during run time, independent from the main program. Use the volatile modifier to indicate that a variable can be changed by a background routine, an interrupt routine or I/O port. Declaring an object to be volatile warns the compiler not to make assumptions concerning the value of an object while evaluating expressions in which it occurs, because the value could be changed at any moment. In the following example, variable *Cnt* is declared to be a volatile unsigned character:

```
volatile unsigned char Cnt;
```

3.2.10 Accessing Bits of a Variable

There are many cases where we may want to access individual bits of an 8-bit variable. If we wish to access the bit of a microcontroller internal register, and if we know the name of the register to be accessed, then we can simply write the name of the bit and set or reset it as required. An example is given below:

```
GIE_bit = 0;                //Clear GIE bit
```

We can also use the qualifiers B0, B1, ... B7 or F0, F1, ... F7, with '0' being the least significant bit (LSB) and '7' being the most significant bit (MSB). As an example, to set bit 0 of register INTCON we can write:

```
INTCON.B0 = 1;            //Set bit 0 of register INTCON
```

or, to set bit 3 we can write:

```
INTCON.F3 = 1;            //Set bit 3 of register INTCON
```

www.it-ebooks.info

3.2.11 *sbit Type*

The mikroC Pro for PIC compiler has *sbit* data type, which provides access to bit addressable internal registers (SFRs). You can access bits of internal registers, as in the following examples:

```
sbit LEDA at PORTA.B0;        //LEDA is assigned to bit 0 of PORT A
sbit LEDB at PORTA.B7;        //LEDB is assigned to bit 7 of PORT A
```

or, alternatively and equivalently,

```
sbit LEDA at RA0_bit;        //LEDA is assigned to bit 0 of PORT A
sbit LEDB at RA7_bit;        //LEDB is assigned to bit 7 of PORT A
```

3.2.12 *bit Type*

mikroC Pro for PIC compiler also supports a single-bit definition using the *bit* type. An example is given below:

```
bit xf;
```

3.2.13 *Arrays*

3.2.13.1 Numeric Arrays

During program development, there is usually the need to manipulate several related items of data of the same type. For example, a program designed to read the ages of 50 students in a classroom may at first seem to require the use of 50 separate integer type variables. However, such an approach makes the manipulation of data difficult, as we have to access each integer separately. Furthermore, if the class size increases to, say, 70 students, then we have to introduce 20 more integers and 20 more statements to process the new entries.

The solution to this problem is to use an array with an easy way of collecting related items under a single variable name. An array is declared by specifying its name, type and the number of elements it has to store. For example, the following is an unsigned integer array called *Average*, having five elements:

```
unsigned int Average[5];
```

The array is stored in sequential memory locations, as shown below:

Average[0]
Average[1]
Average[2]
Average[3]
Average[4]

www.it-ebooks.info

In this example, the first array element has index 0 and the last one has index 4. Array elements are addressed by writing the array name followed by a square bracket where the index is specified. For example, to set the third array element to value 120, we have to write:

```
Average[2] = 120;
```

Similarly, for example, to copy the third array element to a variable called *MyValue*, we can write:

```
MyValue = Average[2];
```

Whenever an array definition is encountered by the compiler, a calculation is performed to determine the storage requirements of each element.

The contents of an array can be initialised during the declaration of the array by specifying the array elements, separated by commas and enclosed in curly brackets. An example follows where array numbers have 10 elements and numbers[0] = 0, numbers[1] = 1, and so on:

```
unsigned int numbers[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

The same array can be declared, without specifying the array size, as follows. Here, the compiler determines the array size and allocates the required number of bytes in memory:

```
unsigned int numbers[ ] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Note that it is not necessary to initialise all of the elements of an array. For example, we could pre-initialise just the first three elements of an array and leave the remaining elements un-initialised with a definition of the form:

```
int MyNumbers[10] = {0, 2, 5};
```

In which case only elements [0], [1] and [2] would be initialised, even though storage will be reserved for 10 elements. Here, any remaining un-initialised elements are implicitly initialised to zero.

Note, however, that it is an error to have more elements than the defined array size. For example, the definition below will generate a compiler error:

```
int numbers[2] = {0, 3, 7, 8, 9};
```

Whenever an array is initialised, the compiler copies the specified data to the array elements. If this array happens to be inside a function, then each time the function is called, the array elements will be re-loaded, thus causing unnecessary delay. For this reason, the keyword *static* should be used to force the array elements to be loaded only once at the start of the program:

```
static int numbers[2] = {0, 5};
```

In mikroC Pro for PIC language, we can also declare an array with multiple dimensions. Such arrays are commonly used in mathematical operations, such as vector and matrix calculations. A multi-dimensional array is declared by specifying the data type, name of the array

www.it-ebooks.info

and the size of each dimension. In the following example, a two-dimensional array called *MyMatrix* is declared, having three rows and two columns:

```
int MyMatrix[3][2];
```

This array will have the following structure. Altogether the array has six elements. The first element of the array is MyMatrix[0][0], and the last element is MyMatrix[2][1]:

MyMatrix[0][0]	MyMatrix[0][1]
MyMatrix[1][0]	MyMatrix[1][1]
MyMatrix[2][0]	MyMatrix[2][1]

Elements of a multi-dimensional array can be initialised as before, by specifying the elements within curly brackets and separated by commas. An example is given below:

```
int numbers[2][3] = { {0, 2, 5}, {6, 8, 5} };
```

In the above example, we have 2 rows and 3 columns. The value of each element can be shown as:

0	2	5
6	8	5

The size of the first dimension is optional and can be left blank, as shown below for the above example. The compiler fills in the correct size during compilation:

```
int numbers[ ][3] = { { 0, 2, 5}, {6, 8, 5} };
```

3.2.13.2 Character Arrays

Character arrays are declared similarly to numeric arrays, where each character is separated and enclosed within a pair of curly brackets. In the following example, character array *MyName* has four elements:

```
unsigned char MyName[4] = {'J', 'O', 'H', 'N'};
```

As before, we can leave the array size blank:

```
unsigned char MyName[ ] = {'J', 'O', 'H', 'N'};
```

3.2.13.3 Strings

Strings are character arrays terminated with a NULL character (hexadecimal 0×0 or '\0'). In the example below, *MyName* is a string having five elements, including the string terminator NULL character:

www.it-ebooks.info

```
unsigned char MyName[ ] = { 'J', 'O', 'H', 'N', '\0' };
```

As you can see from the above example, we have to separate each character with a comma and terminate the string with a NULL character. An alternative and easier way of declaring the same string would be:

```
unsigned char MyName[ ] = "JOHN";
```

Here, the characters are terminated automatically with a NULL character and this second option is more readable, especially when it is required to declare long strings. In addition, the string will not be terminated correctly if we forget to insert the NULL character.

3.2.13.4 Constant Strings

In many applications it may be required to create fixed long strings in the flash program memory of the microcontroller. Such strings can be created as constant strings to save space in the RAM data memory. An example is given below:

```
const unsigned char Text[ ] = "This is the main menu of the program";
```

3.2.13.5 Arrays of Strings

There are many applications where we may want to create arrays of strings in our programs. In the following example, the array *Days* stores days of the week. Notice that it is optional to specify size of the first dimension:

```
char Days[ ][10] = { "Monday",
                     "Tuesday",
                     "Wednesday",
                     "Thursday",
                     "Friday",
                     "Saturday",
                     "Sunday"
                   };
```

In the above example, the size of the first dimension is set to 7 automatically by the compiler. The second dimension is set to 10, which is the size of the longest word in the array. Notice that each word in the array is a string and as such is terminated with a NULL character. Figure 3.2 shows the structure of this array.

3.2.14 Pointers

Pointers are a very important part of the C language, and the subject of pointers is perhaps the most interesting and useful aspect of C. The concept of pointers may sound strange to most students who have been programming in other high-level languages, such as Pascal or BASIC. Pointers are important, especially in microcontroller based applications, since they enable the programmer to directly access the memory locations by using memory addresses.

www.it-ebooks.info

Days[0]	M	o	n	d	a	y	'\0'			
Days[1]	T	u	e	s	d	a	y	'\0'		
Days[2]	W	e	d	n	e	s	d	a	Y	'0\'
Days[3]	T	h	u	r	s	d	a	Y	'0\'	
Days[4]	F	r	i	d	a	Y	'0\'			
Days[5]	S	a	t	u	r	d	a	Y	'0\'	
Days[6]	S	u	n	d	a	y	'0\'			

Figure 3.2 Structure of the array of strings

Pointers hold the addresses of variables in memory. They are declared just like the other variables, but with the character '∗' inserted in front of the variable name. Pointers can be created to point (or hold the address of) to character variables, integer variables, long variables, floating point variables, and so on. Because of this generality, we have to specify the type of a pointer at the time of declaring it.

In the following example, *ptr* is the pointer to a character variable in memory. At this point all we know is that it is a pointer can hold the address of a character variable. But we have not specified yet which variable's address it is holding:

```
unsigned char *ptr;
```

We can now specify the name of the variable whose address we wish to hold. This is done using the character '&' in front of the variable name. In the following example, *ptr* holds the address of character variable *Cnt* in memory:

```
ptr = &Cnt;
```

The value of variable *Cnt* can be accessed by using the '∗' character in front of its pointer. Thus, the following two statements are equivalent:

```
Cnt = 5;          //Cnt = 5
*ptr = 5;         //Set the value of the variable pointed to by ptr to 5
```

We can also make an assignment of the form shown below, to assign value to a variable:

```
Count = *ptr;         //Count = Cnt
```

3.2.14.1 Pointer Arithmetic

In C language we can perform various pointer arithmetic, which may involve:

- adding or subtracting pointers with integer values;
- adding or subtracting two pointers;
- comparing two pointers;
- comparing a pointer to a NULL;
- assigning one pointer to another.

www.it-ebooks.info

1000	A = 23
1001	B = 0
1002	C = 4
1003	D = 100
1004	E = 250
1005	F = 65

Figure 3.3 Memory locations

As an example to pointer arithmetic, assume that six memory locations, starting from address 1000, store the character variables, as shown in Figure 3.3.

We can now declare a pointer to hold the address of variable *A* and then perform the following operations:

```
char *ptr;              //ptr is a character pointer
ptr = &A;               //ptr holds address 1000
ptr = ptr +2;           //ptr now points to 1002
*ptr = 25;              //Variable C = 25
ptr = ptr + 1;          //ptr now points to 1003
E = *ptr;               //Variable E = 100
ptr = ptr + 2;          //ptr now points to 1005
*ptr = 0;               //Variable F = 0
```

3.2.14.2 Array Pointers

In C language, the name of an array is also a pointer to the first element of the array. Thus, for the array:

```
int Sum[10];
```

The name *Sum* is also a pointer to element Sum[0] of the array, and it holds the address of the array. Similarly, the following statements can also be used to point to the array:

```
int *ptr;                   //ptr is an integer pointer
ptr = &Sum[0];              //ptr points to first element of array Sum
```

The following two statements are equivalent, since *Sum* is also the address of array *Sum*:

```
Sum[2] = 0;
*(Sum + 2) = 0;
```

It is interesting to note that the following statement is also true, for the same reason:

```
&Sum[3] = Sum + 3;
```

www.it-ebooks.info

3.2.14.3 Using Pointers in String Operations

Another useful application of pointers is to create and manipulate string variables. Remember that strings are a collection of character arrays terminated with the NULL character. Using pointers, we can create a string, as shown in the example below:

```
char *p = "JOHN";
```

Here, a hidden character array is created at compile time, containing characters 'JOHN', terminated with the NULL character. The character pointer *p* is initialised and loaded with the address of this string in memory. Thus, *p* holds the address of the first character of the string, that is the address of character 'J'. The programmer has no control over the size of the created string and therefore should not attempt to alter its contents using the pointer.

In many applications we may want to create long fixed strings. This is easily done using pointers and declaring the strings as constants. An example is given below:

```
const char *p1 = "My very long text string";
```

It is important to realise the differences between the following two ways of creating strings:

```
cont char Text[ ] = "An example text";
const char *p = "An example text";
```

In the first statement, a character array called *Text* is created and is terminated with a NULL character. Individual characters of this array can be accessed by indexing the array. In the second statement, the characters 'An example text' and terminator NULL are stored somewhere in memory, and pointer *p* is loaded with the address of the first character of this text, that is the address of character 'A'.

We can create arrays of text strings using pointers. In the following example, seven pointers are created with names Days[0] to Days[6] and each pointer is loaded with the corresponding address of the day name. The dimension of the pointer array is set to 7 automatically by the compiler:

```
char *Days[ ] ={    "Monday",
                    "Tuesday",
                    "Wednesday",
                    "Thursday",
                    "Friday",
                    "Saturday",
                    "Sunday"
               };
```

3.2.15 Structures

Structures are used in programs to group and manipulate related but different types of data as a single object or variable. The elements of a structure can all be of different type, which is in contrast with an array where all elements must be of the same type. For example, a structure

www.it-ebooks.info

can be used to store details of a person, such as the name, age, height and weight as a single object. Here, the elements are of different types. Name is a string, age is an integer, and height and weight are floating point type variables.

A structure is created using the keyword struct, followed by a structure name and a list of the elements (or structure members), enclosed within a pair of curly brackets. As an example, the structure called *Person* given below could be used to describe details of a person:

```
struct Person
{
        char name[30];
        char surname[30];
        char address[80];
        int age;
        float height;
        float weight;
};
```

When a structure, as above, is declared, it is simply a template and no space is reserved in memory for this template. It is only when variables of the same type are declared that the structure takes space in memory. For example, we can create a variable called *Me* of type *Person* by the statement:

```
struct Person Me;
```

It is important to realise that it is *Me* that is the variable and not the *Person*. We can create more than one variable of type *Person* by separating them with a comma, as shown below:

```
struct Person Me, You, He, She;
```

In the above examples, the structure template and variable creation were done in two separate statements. If required, these two statements can be combined into one and the variables can be created during the creation of the template. An example is given below:

```
struct Person
{
        char name[30];
        char surname[30];
        char address[80];
        int age;
        float height;
        float weight;
}Me, You, He, She;
```

We can omit the name of the structure if we wish, but if we do this, then all variables based upon that structure template must be defined at the same time as the template is declared, as there is no way to introduce new variables later using the same template.

www.it-ebooks.info

3.2.15.1 Accessing Structure Members

After a structure variable has been defined, its members (elements within the structure) can be individually accessed using the dot ('.') operator. This operator is placed between the structure name and the member name that we wish to access. In the following statement, *age* member of structure *Me* above is set to 25:

```
Me.age = 25;
```

Similarly, for example, the *height* in the same structure can be assigned to a variable called *H* with the statement:

```
H = Me.height;
```

3.2.15.2 Initialising Structure Members

As with other data types, when a structure is declared, the values of its members are undefined. However, as with arrays, it is often necessary to initialise the member elements to known values during the declaration of a structure. The following example shows how the sides of a rectangle declared as a structure can be initialised to 2.5 and 4.0:

```
struct Rectangle
{
        float sideA;
        float sideB;
} MyRectangle = {2.5, 4.0};
```

Members of a structure can also be assigned values by using structure pointers. For example, in the above example, we can define *MyRectangle* as a pointer and then assign values to its members using the arrow ('->') operator:

```
struct Rectangle
{
        float sideA;
        float sideB;
} *MyRectangle;

MyRectangle -> sideA = 2.5;
MyRectangle -> sideB = 4.0;
```

3.2.15.3 Structure Copying

If two structure are derived from the same template, then it is permissible to assign one structure to the other one (this is not possible in arrays). An example is given below:

```
struct Rectangle
{
        float sideA;            //Member sideA of the structure
        float sideB;        //Member sideB of the structure
```

www.it-ebooks.info

```
} R1, R2;              //R1 and R2 are two variables

R1.sideA = 5.0;        //Initialize sideA of R1
R1.sideB = 6.5;        //Initialize sideB of R1

R2 = R1;               //Copy all members of R1 to R2
```

3.2.15.4 Size of a Structure

Sometimes we need to know the size of a structure variable. We can use the *sizeof* operator to find the number of bytes occupied by a structure variable. In the following example, the size of structure variable *R1* is assigned to integer variable *S*:

```
S = sizeof(R1)
```

3.2.15.5 Arrays of Structures

In some applications we may need a collection of structures of similar type. For example, consider the following complex number structure template:

```
struct complex_number
{
        float Real_Part;
        float Imaginary_Part;
}
```

We can now create an array of this structure with 10 elements as:

```
struct complex_number XY[10];
```

Individual structure elements can now be accessed by indexing the structure variable *XY*. In the example below, index 0 of the structure variable is taken and the real and imaginary parts are set to 2.3 and 5.0, respectively:

```
XY[0].Real_Part = 2.3;
XY[0].Imaginary_Part = 5.0;
```

3.2.15.6 Structure Bit Fields

Bit fields can be defined using structures. With bit fields, we can assign identifiers to individual bits or to collections of bits of a variable. For example, to identify the low byte of a 16-bit unsigned integer *Total* as *LowB* and the high byte as *HighB*, we can write:

```
struct
{
        LowB: 8;
        HighB: 8;
} Total;
```

www.it-ebooks.info

We can then access the two bytes as:

```
Total.LowB = 250;
Total.HighB = 125;
```

3.2.16 Unions

A union in C is very much like a structure, and is even declared and initialised in the same way. Both are based on templates and members of both are accessed in the same way. Where a union principally differs from a structure is that in a union all members share the same common storage area. Even though the members can be of different type, they all share the same common storage area, and the size of this area is equal to the size of the largest data type amongst the members of the union. An example of a union declaration is:

```
union Temp
{
        unsigned int x;
        unsigned int y;
        unsigned char z;
} V;
```

In this example, variables *x*, *y* and *z* all share the same memory area. Variables *x* and *y* are mapped to each other, where variable *z* is mapped to the lower byte of *x* or *y*. The size of the common area in this example is 2 bytes. In the following statement, 16-bit hexadecimal number 0×F034 is loaded into union variable *V*:

```
V.x = 0xF034;
```

Now, both variables *x* and *y* are loaded with hexadecimal number 0×F034, and variable *z* is loaded with the lower byte, that is 0×34.

As with the structures, the size of a union variable can be determined using the size of operator, as shown below:

```
S = sizeof(V);                    //Returns 2
```

3.2.17 Operators in mikroC Pro for PIC

Operators are symbols applied to variables to cause some operations to take place. For example, addition symbol '+' is an operator and causes the value of a variable to change. In mikroC Pro for PIC language, operators are classed as unary or binary. Unary operators require only one variable and they operate on this variable, for example changing the sign of a variable. Binary operators, on the other hand, operate on two variables, for example adding two numbers.

Operators in mikroC Pro for PIC can be arithmetical, logical, bitwise, relational, assignment, conditional and pre-processor. In this section, we shall look at these operators in detail and see how they can be used in programs.

www.it-ebooks.info

Table 3.4 mikroC Pro for PIC arithmetic operators

Operator	Operation
+	Addition
−	Subtraction
*	Multiplication
/	Division
%	Remainder (integer division)
++	Auto increment
--	Auto decrement

3.2.17.1 Arithmetic Operators

Table 3.4 gives a list of the arithmetical operators. All readers are familiar with the basic operators of addition, subtraction and multiplication. The other arithmetic, operators need some explanation and some understanding before they are used.

The division operator '/' divides two numbers. If the numbers are real, then using floating point arithmetic will give correct results. But, if the two numbers are integers, the division can give wrong results, as in the following example:

```
int x, y, z;
x = 3;
y = 4;
z = x/y;                    //The result is 0
```

The modulus operator '%' is used to give the remainder after two integer numbers are divided. In the following example, numbers 5 and 7 are divided using the modulus operator and the result is 2, which is the remainder:

```
int x, y, z;
x = 5;
y = 7;
z = 7 % 5;                   //z = 2 (the remainder)
```

The auto increment operator (++) is used to increment the value of a variable by 1, without using the assignment operator ('='). An example is given below:

```
int x;
x = 5;
x++;                         //x = 6
```

Similarly, the auto decrement operator (--) is used to decrement the value of a variable by 1, without using the assignment operator. An example is given below:

```
int x;
x = 8;
x-;                         //x = 7
```

www.it-ebooks.info

The auto increment and auto decrement operators can be used in assignment operations. The value assigned to a variable changes depends on whether the '++' or '--' symbols are placed to the left or the right of a variable. In the following example, the value of variable *Sum* is 10 initially. *Sum* is assigned to variable *Total* and is then incremented automatically. Thus, at the end, *Sum* contains 11 and *Total* contains 10:

```
int Sum, Total;
Sum = 10;
Total = Sum++;            //Total = 10, Sum = 11
```

In the following example, *Sum* is incremented and is then assigned to *Total*. Thus, at the end, *Sum* contains 11 and *Total* contains 11:

```
int Sum, Total;
Sum = 10;
Total = ++Sum;            //Total = 11; Sum = 11
```

A similar thing happens when auto decrement is used. An example is given below:

```
int Sum, Total;
Sum = 10;
Total = Sum--;              //Total 10, Sum = 9
Total = --Sum;              //Total = 8, Sum = 8
```

3.2.17.2 Logical Operators

Logical operators are used in logical and arithmetical operations. Table 3.5 gives a list of the mikroC Pro for PIC logical operators.

An example is given below on the use of logical operators:

```
Assume x = 10, y = -2

x > 0 && y < 0              //Returns TRUE (1)
x > 0                       //Returns TRUE (1)
y < 0                       //Returns TRUE (1)
x > 0 || y < 0              //Returns TRUE (1)
x < 0                       //Returns FALSE (0)
```

Table 3.5 mikroC Pro for PIC logical operators

Operator	Operation
&&	AND
||	OR
!	NOT

www.it-ebooks.info

Table 3.6 mikroC Pro for PIC bitwise operators

Operator	Operation
&	Bitwise AND
\| | Bitwise OR |
^ | Bitwise EXOR |
~ | Bitwise complement |
<< | Shift left |
>> | Shift right |

3.2.17.3 Bitwise Operators

In addition to arithmetical and logical operators, bitwise operators are used to modify individual bits of a variable. Table 3.6 gives a list of the mikroC Pro for PIC bitwise operators. These operators can only be used with integer type variables.

&

The '&' operator performs the bitwise AND of its two operands. Each bit of the first operand is ANDed with the corresponding bit of the second operand. The resulting bit is only 1 if both of the two corresponding bits are 1, otherwise the resulting bit is set to 0. An example is given below:

```
int x, y, z;
x = 0xF0E0;              //x = "1111 0000 1110 0000"
y = 0x0F71;              //y = "0000 1111 0111 0001"
z = x & y;               //z = "0000 0000 0110 0000"
```

In this example, variable z takes the hexadecimal value 0×0060. The bitwise ANDing operation is shown below:

```
x: 1111 0000 1110 0000
y: 0000 1111 0111 0001
   ------------------
z: 0000 0000 0110 0000 i.e. 0x0060
```

|

The '|' operator performs the bitwise OR of its two operands. Each bit of the first operand is Ored with the corresponding bit of the second operand. The resulting bit is 1 if either of the corresponding bits are 1, otherwise the resulting bit is set to 0. An example is given below:

```
int x, y, z;
x = 0xF0E0;              //x = "1111 0000 1110 0000"
y = 0x0F71;              //y = "0000 1111 0111 0001"
z = x | y;               //z = "1111 1111 1111 0001"
```

www.it-ebooks.info

In this example, variable *z* takes the hexadecimal value 0×FFF1. The bitwise ORing operation is shown below:

```
x: 1111 0000 1110 0000
y: 0000 1111 0111 0001
   ------------------
z: 1111 1111 1111 0001 i.e. 0xFFF1
```

^

The '^' operator performs the bitwise Exclusive OR of its two operands. Each bit of the first operand is Exclusive Ored with the corresponding bit of the second operand. The resulting bit is 1 if only one of the corresponding bits is 1, otherwise the resulting bit is set 0. An example is given below:

```
int x, y, z;
x = 0xF0E0;                 //x = "1111 0000 1110 0000"
y = 0x0F71;                 //y = "0000 1111 0111 0001"
z = x ^ y;                  //z = "1111 1111 1001 0001"
```

In this example, variable *z* takes the hexadecimal value 0×FF91. The bitwise Exclusive ORing operation is shown below:

```
x: 1111 0000 1110 0000
y: 0000 1111 0111 0001
   ------------------
z: 1111 1111 1001 0001 i.e. 0xFF91
```

~

The '∼' operator performs the bitwise complement of its operand. Each bit is complemented, thus a 0 becomes a 1, and a 1 becomes a 0. An example is given below:

```
int x, y;
x = 0xF0E0;                 //x = "1111 0000 1110 0000"
y = ~x  ;                   //y = "0000 1111 0001 1111"
```

In this example, variable *y* takes the hexadecimal value 0×0F1F. The bitwise complement operation is shown below:

```
x: 1111 0000 1110 0000
  ------------------
y: 0000 1111 0001 1111 i.e. 0x0F1F
```

<<

The '<<' operator performs shift left. The operand is shifted left by n bits, where n is specified by the programmer. The right-hand side of the variable LSB is filled with zeroes,

www.it-ebooks.info

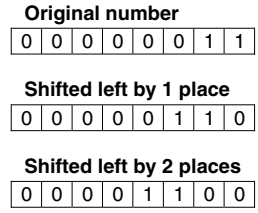


Figure 3.4 Shifting left by 2 places

while bits at the left-hand side MSB are lost. Shifting a variable left by 1 bit is the same as multiplying by 2, but the shift operation is much quicker. An example is given below:

```
int x, y;
x = 3;
y = x << 2;                    //y = 12
```

The actual shift left operation for the above example is shown in Figure 3.4.

>>

The '>>' operator performs shift right. The operand is shifted right by n bits, where n is specified by the programmer. The left-hand side of the variable MSB is filled with zeroes, while bits at the right-hand side LSB are lost. Shifting a variable right by 1 bit is the same as dividing by 2, but the shift operation is much quicker. An example is given below:

```
int x, y;
x = 12;
y = x >> 2;                    //y = 3
```

The actual shift right operation for the above example is shown in Figure 3.5.

3.2.17.4 Relational Operators

Relational operators are used in conditional operations that change the flow of control in a program. A 1 is returned if the relational operation evaluates to TRUE, otherwise 0 is

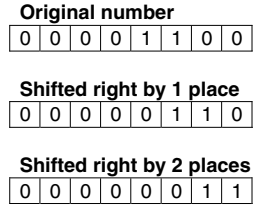


Figure 3.5 Shifting right by 2 places

www.it-ebooks.info

Table 3.7 microC relational operators

Operator	Operation
= =	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

returned. Table 3.7 gives a list of the mikroC relational operators. These operators are common to all programming languages, but the symbols used may change. It is important to note that the equivalence operator '= =' consists of two equal signs. A common mistake is to use a single equal sign in a relational operator, which will be interpreted as an assignment operator.

A few examples to the use of relational operators are given below:

```
x = 20;
x > 10;          //Returns 1 (TRUE)
x <= 0;          //Returns 0 (FALSE)
x != 5;          //Returns 1 (TRUE)
```

3.2.17.5 Assignment Operator

The assignment operator '=' is used to assign the result of an expression to a variable. The general format of the assignment operation is:

```
variable = expression;
```

In the following example, the sum of *a* and *b* are assigned to *c*:

```
c = a + b;
```

mikroC Pro for PIC also supports complex assignment operations, used when a variable appears both on the left and right of the assignment operator. For example, consider the expression below, where variable *a* is added to variable *b* and the result is stored back in variable *a*:

```
a = a + b;
```

Using complex assignment operator, we can write the above statement as:

```
a += b;
```

www.it-ebooks.info

Other useful assignment operators are:

```
-=      subtraction
*=      multiplication
/=      division
&=      AND
|=      OR
>>=     right shift
<<=     left shift
^=      exclusive OR
```

3.2.17.6 The Conditional Operator

The conditional operator (or ternary operator) is useful for evaluating conditional expressions. This operator requires three operands and its general form is given below and is interpreted as follows: if *expression1* is TRUE, then evaluate *expression2* and assign to the *result*, otherwise evaluate *expression 3* and assign to the *result*.

```
result = expression1 ? expression2:expression3;
```

In the following example, the minimum of *x* and *y* is found and is assigned to *min*. Notice that using brackets simplifies the expression:

```
min = (x < y) ? x:y;
```

In the following example, the absolute value of *x* is assigned to *y*. We can interpret this expression as: if *x* is less than zero (i.e. negative) then change its sign (i.e. make it positive) and assign to *y*, otherwise assign *x* to *y* (since it is already positive)

```
y = (x < 0) ? -x:x;
```

3.2.17.7 Preprocessor Operators

Any line in the source code with a leading '#' character is known as a preprocessor operator, or preprocessor directive. Traditionally, the preprocessor has been a separate program, which runs before the main compilation process. However, with modern compilers, the preprocessor is part of the main compiler and is automatically invoked at compilation time.

Preprocessor operators are useful when the programmer wants to insert files into a program, when it is required to replace symbols or values with other symbols, or when it is required to compile part of a program conditionally.

mikroC Pro for PIC supports the following preprocessor operators:

```
# (null)          #if
#define           #ifdef
#elif             #ifndef
```

www.it-ebooks.info

```
#else        #include
#endif       #line
#error       #undef
```

#define

The #define preprocessor operator is used at the beginning of a program and it is useful when we wish to assign symbols to values. Some examples are given blow:

```
#define PI       3.14159
#define MAX      2000
#define GT       >
#define MIN      0
```

It is important to realise that when the #define preprocessor operator is used, the compiler substitutes the value for the symbol wherever it is used in the program. Also, an identifier that has already been defined cannot be defined again. One way to get round this problem is to un-define a definition using preprocessor operator #undef, as shown below:

```
#undef PI
```

Alternatively, we can use the existence of a definition using the #ifndef preprocessor operator, as shown below. Here, the definition of symbol PI is checked for existence. If it is not defined, a definition is made, otherwise nothing else is done:

```
#ifndef PI
        #define PI 3.14159
#endif
```

The #define preprocessor operator may include a number of optional parameters. An example is given below:

```
#define SQR(x)     ((x)*(x))
```

When symbol SQR is used in a program, (x) will be replaced with the square of it, that is ((x)∗(x)). Thus, the following expression

```
a = SQR(b);
```

will evaluate into

```
a = ((b)*(b));
```

Some other examples are given below:

```
#define SUM(a,b)    ((a) + (b))
#define CUBE(x)     ((x) * (SQR(x)))
```

www.it-ebooks.info

It is important to enclose the definitions on the right-hand side within the parenthesis for the expressions to be expanded correctly. As an example, consider what may happen if parenthesis is not used:

```
#define SQR(x)    x*x
```

Now, suppose we use this definition in a statement such as

```
p = SQR(2 + 4)/4;
```

Now, the compiler will expand the above expression as

```
p = 2 + 4 * 2 + 4/4;
```

giving the wrong result 11 instead of the correct result of 9.

#include

The #include preprocessor operator is used to insert a file into the program. It is commonly used at the beginning of a program to insert a header or a definition file into the program. In the following example, file math.h is inserted into the program:

```
#include <math.h>
```

or

```
#include "math.h"
```

If the #include is used with the '< >' version, the compiler searches the file in each of the following locations, in the given order:

- the mikroC Pro for PIC installation folder 'include';
- user's custom search path.

The ' ' version searches the file in the following locations, in the given order:

- user project folder;
- mikroC Pro for PIC installation folder 'include';
- user's custom search path.

It is also permissible to specify the search path explicitly, as shown in the following example:

```
#include "C:\My_Folder\math.h"
```

#if, #elif, #else, #endif

www.it-ebooks.info

These are conditional assembly preprocessor operators. Using these operators, the programmer can force parts of a program to be compiled conditionally. For example, depending on the type of microcontroller used, the programmer may wish to not compile part of a program.

In the following example, the code section including variable *FLAG* is compiled if *CPU* is 100, otherwise this code section is omitted:

```
#if CPU == 100
      FLAG = 1;
#endif
```

Similarly, in the following example, if *CLOCK* is 1 the first code block is compiled, otherwise the second code block is compiled:

```
#if CLOCK == 1
      A = 10;
      B = 20;
#else
      A = 1;
      B = 2;
#endif
```

We can also use the #elif in conditional compilation, as shown in the following example. Here, if *Y_RES* is less than 500, then the code X = 200 is compiled, if *Y_RES* is between 500 and 1024, then the code X = 300 is compiled, if *Y_RES* is greater than 1024, then the code X = 400 is compiled, otherwise the code X = 500 is compiled:

```
#if Y_RES < 500
        X = 200;
#elif Y_RES >= 500 && Y_RES < 1024
        X = 300;
#elif Y_RES >= 1024
        X = 400;
#else
        X = 500;
#endif
```

Notice that each #if operator must match with a #endif operator. Any number of #elif operators can be used between #if and #endif, but at most only one #else can be used.

3.2.18 The Flow of Control

Most statements in a program are executed sequentially one after the other. There are many cases where in a program a decision is to be made about the operation that will be performed next. The flow of control in a program can be modified using the flow of control statements. These statements include modification of flow based on selection,

www.it-ebooks.info

unconditional modification of the flow, and iteration statements. In this section, we shall be looking at these statements that modify the flow of control in programs.

3.2.18.1 Selection Statements

The selection statements are based on modifying the flow of control conditionally. For example, if a condition evaluates to TRUE, then a certain part of the program is executed, otherwise a different part is executed.

There are two selection statements: *if - else* and *switch*.

if – else Selection Statements

The basic *if* statement has the following format, where the expression usually includes logical and relational operators:

```
if(expression) statement;
```

In the following example, if variable *p* is equal to 10 then *k* is incremented by 1:

```
if(p == 10) k++;
```

Notice that because the white spaces are ignored in the C language, the statement can be on a different line:

```
if(p == 10)
       k++;
```

The *if* statement can be used together with *else*, such that if the expression is FALSE, then the statement following *else* is executed. In the following example, if variable *Total* is equal to *MAX*, then *Sum* is incremented by 1, otherwise *Sum* is decremented by 1:

```
if(Total == MAX) Sum++; else Sum-;
```

The above statement is usually written in the following form:

```
if(Total == MAX)
         Sum++;
else
         Sum-;
```

In some applications, we may need to execute more than one statement if a condition is TRUE or FALSE. This is done by enclosing such statements inside a pair of curly brackets. An example is given below:

```
if(Cnt > Sum)
{
    i = k + 4;
    j = j*2;
}
```

www.it-ebooks.info

or

```
if(i == 10 && Cnt < 100)
{
    r1 = 2*x;
    r2 = i + 4;
}
else
{
    r1 = 0;
    r2 = 0;
}
```

3.2.18.2 switch Selection Statements

There are cases where we may want to do multi-way conditional tests. For example, in a menu type application, the user's choice is compared to a number of options, and a different action is taken with each option. Such a program could normally be written using the *if – else* type selection statements. However, when there are many options, the use of *if – else* makes the program unreadable.

The switch statement is generally used in multi-way conditional tests. The general format of this statement is:

```
switch (condition)
{
        case condition 1:
                statements;
                break;
        case condition 2:
                statements;
                break;
        ..................
        ..................
        case condition n:
                statements;
                break;
        default:
                statements;
                break;
}
```

The switch statement operates as follows: First, the *condition* is evaluated. If *condition* is equal to *condition 1*, then statements following *condition 1* are executed. If *condition* is equal to *condition 2*, then statements following *condition 2* are executed. This process continues until *condition n*, where if the *condition* is equal to *condition n*, then statements following *condition n* are executed. If *condition* is not equal to any of the *conditions 1* to *n*, then

www.it-ebooks.info

the statements following the *default* case are executed. Notice that the *break* statements in each block make sure that we return to the end of the *switch* statements (after the closing curly bracket) and jump out of the selection block. Without the *break* statements, the program will continue to execute those statements associated with the next case. The last *break* statement is not obligatory, as the switch block closes after this statement. The *default* case is optional and can be avoided if we are sure that the *condition* will be equal to one of the *conditions 1* to *n*.

The following example shows part of a program implemented using both *if-else* and *switch* selection statements. Functionally, the two codes are equal to each other:

```
if(x == 1)
        statement1;
else if(x == 2)
        statement2;
else if(x == 3)
        statement3;
else
        statement4;
```

```
switch(x)
      {
       case 1:
              statement1;
              break;
       case 2:
              statement2;
              break;
       case 3:
              statement3;
              break;
       default:
              statement4;
  }
```

An example is given below, which shows how the *switch* selection statement can be used.

Example 3.1

It is required to write a program code using the *switch* statement to convert a hexadecimal character A to F to its decimal equivalent. Assume that the input character is stored in a variable called *chr*, and the output should be stored in variable *y*.

Solution 3.1

As you will remember, the decimal equivalents of hexadecimal digits are:

Hexadecimal	Decimal
A	10
B	11
C	12
D	13
E	14
F	15

www.it-ebooks.info

The required program code is given below:

```
switch (chr)
{
    case 'A':
          y = 10;
          break;
    case 'B':
          y = 11;
          break;
    case 'C':
          y = 12;
          break;
    case 'D':
          y = 13;
          break;
    case 'E':
          y = 14;
          break;
    case 'F':
          y = 15;
          break;
}
```

Example 3.2

The relationship between *X* and *Y* variables in an experiment are as follows:

X	*Y*
1	1.5
3	3.5
5	4.0
7	5.0
9	7.5

Write a *switch* statement that will return the value of *Y*.

Solution 3.2

The required *switch* statement is:

```
switch (Y)
{
    case 1:
          X = 1.5;
          break;
```

www.it-ebooks.info

```
    case: 3
          X = 3.5;
          break;
    case 5:
          Y = 4.0;
          break;
    case 7:
          X = 5.0;
          break;
    case 9:
          X = 7.5;
          break;
}
```

3.2.18.3 Repetition Statements

Another fundamental building block in any programming language is the repetition, or looping construct. Here, one or a group of instructions are executed repeatedly, until some condition is satisfied. Basically, mikroC Pro for PIC language supports three types of repetition statements: *while*, *do – while* and *for* loops.

3.2.18.4 while Statements

The general format of the *while* statement is:

```
while(expression) statement;
```

Here, the statement is executed as long as the expression is TRUE. In general, the number of statements are more than one and curly brackets are used to enclose the statements:

```
while(expression)
{
      statements;
}
```

In the following example, the loop formed using the *while* statement is repeated 10 times. The loop control variable is *i* and is initialised to 0 before entering the loop. The statements are executed as long as *i* is less than 10. Notice that inside the loop, the loop counter *i* is incremented by 1 at each iteration:

```
i = 0;
while(i < 10)
{
       statements;
       i++;
}
```

www.it-ebooks.info

Example 3.3

Write a program to initialise an integer array called *MyArray* to zero. Assume that the array has 10 elements.

Solution 3.3

The required program is given below:

```
void main()
{
      unsigned char i;
      int MyArray[10];

     i = 0;
     while(i < 10)
     {
            MyArray[i] = 0;
            i++;
     }
}
```

When using the *while* statement, it is important to ensure that the condition that holds the loop is changed inside the loop, otherwise an infinite loop is formed. The following is the above program written in error that never terminates. The program repeatedly initialises element 0 of the array:

```
void main()
{
      unsigned char i;
      int MyArray[10];

     i = 0;
     while(i < 10)
     {
            MyArray[i] = 0;
     }
}
```

Sometimes it may be necessary to create infinite loops in our programs. This can easily be achieved using the *while* statement, as shown below:

```
while(1)
{
        statements;
}
```

www.it-ebooks.info

3.2.18.5 do – while Statements

This is a useful variation of the *while* statement. The general format of the *do – while* statement is:

```
do
{
        statements;
} while(expression);
```

Here, the statements are executed as long as the expression is TRUE. The expression usually includes logical and relational statements. An example is given below, where the loop is terminated 10 times:

```
i = 0;
do
{
        statement;
        i++;
} while(i < 10);
```

Notice again that the condition that holds the loop is changed inside the loop, so that the loop is terminated after the required number of iterations. There is a difference between the simple *while* and the *do – while* statements. The statements inside the *while* loop may never be executed if the condition is FALSE. On the other hand, the statements inside the *do – while* loop are executed at least once since the condition is tested at the end of the construct. Some examples are given below, showing how the *do – while* loop can be constructed and used in programs:

Example 3.4

Write a program to initialise an integer array called *MyArray* to 1. Assume that the array has 100 elements.

Solution 3.4

The required program is given below:

```
void main()
{
      unsigned char i;
      int MyArray[100];

      i = 0;
      do
      {
            MyArray[i] = 1;
            i++;
      } while(i < 100);
}
```

www.it-ebooks.info

Example 3.5

Write the loop code using *do – while* statement to copy a string called *B* to another string called *A*.

Solution 3.5

Remember that strings are character arrays terminated with a NULL character. The *do – while* loop below will copy the contents of string *B* to *A*, including the NULL terminator. Note that *i* is incremented after its value has been used as a subscript to *B*:

```
i = 0;
do
{
        A[i] = B[i];
} while(B[i++] != '\0');
```

When using the *do – while* statement, it is important to make sure that the condition that holds the loop is changed inside the loop, otherwise an infinite loop is formed. The following is the above program written in error that never terminates. The program repeatedly initialises element 0 of the array:

```
void main()
{
       unsigned char i;
       int MyArray[100];
       i = 0;
       do
       {
            MyArray[i] = 1;
       } while(i < 100);
}
```

Sometimes it may be necessary to create infinite loops. This can easily be achieved using the *do – while* statements, as shown below:

```
do
{
        statements;
} while(1);
```

Example 3.6

Write a program to multiply two integer arrays *X* and *Y* having 10 elements each, and store the sum in an integer variable called *Sum*.

Solution 3.6

The required program is given below:

```
void main()
{
      int X[10], Y[10], Sum;
```

www.it-ebooks.info

```
unsigned char i = 0;
Sum = 0;
do
{
        Sum = Sum + X[i]*Y[i];
        i++;
} while(i < 10);
```

3.2.18.6 for Statements

The *for* statement is the most commonly used statement for setting up loops in programs. The general format of this statement is:

```
for(expression1; expression2; expression3) statements;
```

where *expression 1* sets the initial value of the loop count and this variable is evaluated just once on entry into the loop. *Expression 2* is the condition that keeps the loop continuing. This expression is evaluated at each iteration, to check whether or not the loop should continue. *Expression 2* is usually a logical or relational operator. *Expression 3* is also evaluated at the end of each iteration and this expression changes value that is tested for loop termination. *Expression 3* is usually an increment of the loop counter value.

The *for* loop is commonly used with more than one statement. An example is given below, where the loop is repeated 10 times. The initial value of the loop counter *i* is 0, it is incremented by 1 at each iteration, and the loop continues as long as *i* is less than 10:

```
for(i = 0; i < 10; i++)
{
       statements;
}
```

Some examples of using the *for* statement are given below.

Example 3.7

Write a program to calculate the squares of numbers between 1 and 49 and store the results in an integer array called *Mult*.

Solution 3.7

The required program is given below.

```
void main()
{
      int Mult[50];
      unsigned char i;

      for(i = 1; i <= 49; i++)
      {
            Mult[i] = i*i;
      }
}
```

www.it-ebooks.info

The *for* loops can easily be nested, as shown in the following example. Here, all elements of a two-dimensional array *MyArray* are initialised to 0 (assuming that the array dimension is 10). In this example, the inner loop is repeated 10 times for each *i* value of the outer loop, that is the total number of iterations is 100:

```
for(i = 0; i < 10; i++)
{
        for(j = 0; j < 10; j++)
        {
                MyArray[i][j] = 0;
        }
}
```

The parameters in a *for* loop can be omitted. There are several variations of the *for* loop, depending upon which parameter is omitted. Some examples are given below:

To create an infinite loop:

```
for(;;)
{
        statements;
}
```

Setting the initial loop condition outside the loop:

```
i = 0;
for(; i < 10;)
{
         statements;
         i++;
}
```

To terminate the loop from inside the loop:

```
i = 0;
for(;; i++)
{
    statements;
    if(i >= 10) break;
}
```

3.2.18.7 Premature Termination of a Loop

There are cases where we may want to terminate a loop prematurely. This is done using the *break* statement. When the program encounters this statement, it forces an unconditional jump to outside the loop. The *break* statement can be used with all repetition statements. An

www.it-ebooks.info

example is given above with the *for* statement. In the following example, the loop is terminated when variable *p* becomes 10:

```
p = 0;
while(p < 100)
{
      statements;
      p++;
      if(p == 10) break;
}
```

3.2.18.8 The goto Statement

The *goto* statement can be used to alter the normal flow of control in a program. This statement can either be used on its own or with a condition. The general form of using this statement is:

```
      statements;
Loop:
      statements;
      goto Label;
```

or

```
      statements;
Loop:
      statements;
      if(condition) goto Label;
```

In the first form, the program jumps to the statement just after the label named LOOP. A label can be any valid variable name and it must be terminated with a colon. In the second example, the program jumps to the specified label if a condition is TRUE.

Excessive use of the *goto* statement is not recommended in programs, as it causes a program to be unstructured and difficult to maintain. Very large and complex programs can be written without the use of the *goto* statement. Perhaps the most useful application of the *goto* statement is to prematurely terminate a program.

3.3 Functions in mikroC Pro for PIC

In general, it is a good programming practise to write a large complex program as a collection of smaller modules, where each module can be tested independent of the main code. A function can be thought of as a self-contained, testable program code, developed to perform a specific task. Functions are also created when it is required to repeat a certain algorithm at several different places of the main program. For example, it may be required to convert the temperature from °F into °C at several places in a program. Instead of repeating the code, it is more efficient and the program is more maintainable if a temperature conversion code is

www.it-ebooks.info

written in the form of a function. This function can then be called whenever it is required to make the required conversion.

The general format of a function declaration is as follows:

```
type name(parameters)
{
    body
}
```

Functions usually (not always) perform a certain operation and return data to the calling program. The function type indicates the type of returned data, name is the name of the function, and the parameters (if any) should be separated by commas. The statements inside the function should be enclosed within a pair of curly brackets.

An example function declaration is given below, which calculates the circumference of a circle and returns to the calling program. Here, the radius of the circle is passed as an argument to the function:

```
float Circumference(float radius)
{
     float c;
     c = 2*PI*radius;
     return c;
}
```

Assuming that we wish to calculate the circumference of a circle whose radius is 2.5 cm, and store the result in a variable called *Circ*, the above function can be called, as shown below:

```
Circ = Circumference(2.5);
```

Some more examples are given below.

Example 3.8

Write two functions to calculate the area and volume of a cylinder. Show how these functions can be used in a program to calculate the area and volume of a cylinder whose radius and height are 3.0 and 12.5 cm, respectively. Store the area in variable *c_area*, and the volume in variable *c_volume*.

Solution 3.8

Figure 3.6 shows the two functions. Function *Cyl_Area* calculates the area of a cylinder. Similarly, *Cyl_Volume* calculates the volume of a cylinder.

The main program calling these functions is shown in Figure 3.7. The radius and height of the cylinder are used as parameters to the functions. The area is stored in variable *c_area*, and the volume is stored in variable *c_volume*.

3.3.1 Function Prototypes

If a function is not defined before it is used in the main program, then the compiler generates an error. This is because the compiler does not know what type of data the function returns and the type of its parameters. One way to avoid error messages is to create a function

www.it-ebooks.info

```
//
// This function calculate the area of a cylinder. The radius and height
// are passed as parameters to the function
//
        floatCyl_Area(float radius, float height)
        {
                float c;
                c = 2*PI*radius*height;
                return c;
        }


//
// This function calculate the volume of a cylinder. The radius and height
// are passed as parameters to the function
//
        floatCyl_Volume(float radius, float height)
        {
                float c;
                c = PI*radius*radius*height;
                return c;
        }
```

Figure 3.6 Functions to calculate area and volume of a cylinder

prototype and declare it at the beginning of the program before the function is called by the main program. A function prototype consists of the type and name of the function, followed by the types of its parameters. An example function prototype declaration is given below:

```
float Circumference(float radius);
```

The name of the parameters are optional, and can be avoided as:

```
float Circumference(float);
```

Figure 3.8 shows how the program shown in Figure 3.7 can be modified to use function prototypes. Here, the function prototypes are declared before the main program, and the two functions are declared after the main program, before they are declared.

3.3.2 void Functions

A void function is one where the keyword *void* appears as the function's type specifier, to indicate that the function does not return any value to the calling program. Similarly, a function with no parameters is specified using the keyword *void* in its parameter list. Such a function is called with no parameters. An example void function is shown below, which simply sets all PORT B output pins to a high state. This function has no parameters and does not return any data:

```
void SET_LED(void)
{
      PORTB = 0xFF;
}
```

www.it-ebooks.info

```
/*-------------------------------------------------------------------------------------------

This program calcultes the area and volume of a cylinder. The radius and height
of the cylinder are passed as parameters to two functions which calculate the
area and the volume.

In this example the radius and height are assumed to be 3.0cm and 12.5cm
respectively.

The area is stored in variable c_area, and the volume is stored in variable
c_volume.

Author:      D. Ibrahim
Date:        October, 2011
File:        Cylinder.C
-------------------------------------------------------------------------------------------*/
//
// This function calculates the area of a cylinder. The radius and height are passed
// as parameters to the function
//
      floatCyl_Area(float radius, float height)
      {
            float c;
            c = 2*PI*radius*height;
            return c;
      }

//
// This function calculates the volume of a cylinder. The radius and height are
// passed as parameters to the function
//
      float Cyl_Volume(float radius, float height)
      {
            float c;
            c = PI*radius*radius*height;
            return c;
      }

//
// Main program
//
void main()
{
      float r, h, c_area, c_volume;
      r = 3.0;                                    // The radius
      h = 12.5;                                   // The height
      c_area = Cyl_Area(r, h);                    // Calculate the area
      c_volume = Cyl_Volume(r, h);                // Calculate the volume
}
```

Figure 3.7 Program to calculate the area and volume of a cylinder

3.3.3 Passing Parameters to Functions

It is important to realise that whenever a function is called with parameters (except an array, which is dealt with in the next section), all the parameters are passed to the function by *value*. This means that the values of these parameters are copied to the function and used locally by the function. The values of these parameters cannot be modified by the function. A simple example is given in Figure 3.9, to clarify this concept:

www.it-ebooks.info

```
/*-------------------------------------------------------------------------------------------

This program calcultes the area and volume of a cylinder. The radius and height
of the cylinder are passed as parameters to two functions which calculate the
area and the volume.

In this example the radius and height are assumed to be 3.0cm and 12.5cm
respectively.

The area is stored in variable c_area, and the volume is stored in variable
c_volume.

Author:     D. Ibrahim
Date:       October, 2011
File:       Cylinder.C
-------------------------------------------------------------------------------------------*/
float Cyl_Area(float, float);
float Cyl_Volume(float, float);
//
// Main program
//
void main()
{
        float r, h, c_area, c_volume;
        r = 3.0;                                    // The radius
        h = 12.5;                                   // The height
        c_area = Cyl_Area(r, h);                    // Calculate the area
        c_volume = Cyl_Volume(r, h);                // Calculate the volume
}

//
// This function calculates the area of a cylinder
//
        float Cyl_Area(float radius, float height)
        {
                float c;
                c = 2*PI*radius*height;
                return c;
        }

//
// This function calculates the volume of a cylinder
//
        float Cyl_Volume(float radius, float height)
        {
                float c;
                c = PI*radius*radius*height;
                return c;
        }
```

Figure 3.8 Program in Figure 3.7 modified to use function prototypes

In this example, the value of the function parameter (*a*) was 5 before calling the function. Inside the function, the local copy of this parameter is incremented to 6, but the value of this parameter is still 5 in the main program. When the function returns, variable *b* is assigned to value 6.

www.it-ebooks.info

```
float Inc(float x)
{
        x++;
        return x;
}

//
// Main program
//
void main()
{
        float a, b;
        a = 5.0;                        // a = 5.0
        b = Inc(a);                     // a = 5.0, b = 6.0
}
```

Figure 3.9 Passing parameters by value

3.3.4 *Passing Arrays to Functions*

In some applications, we wish to pass arrays to functions. Passing a single element of an array is easy. All we have to do is index the element that we wish to pass. As described in the previous section, such a variable is passed to a function as a *value*. For example, to pass index 5 of array *MyArray* to a function called *Sum* and store the return value in *a*, we simply write:

```
a = Sum(MyArray[5]);
```

Passing a complete array to a function is slightly more complicated. Here, we simply write the name of the array in the calling program. In the function header we have to declare an array of same type followed with a pair of empty brackets. It is important to realise that we are not copying the entire array to the function, but simply passing the address of the first element of the array, which is also equal to the name of the array. Because the address is passed to the function, the array elements are said to be passed by reference. As a result, the original array elements can be modified inside the function. An example is given below, to illustrate the process.

Example 3.9

Write a program to load an integer array called *N* with values 1 to 10. Then call a function to calculate the sum of the array elements and return the sum to the calling program.

Solution 3.9

The required program listing is given in Figure 3.10. Integer array *N* is initialised with values 1 to 10 in the main program. Then, function *Sum* is called and the array is passed to this function. The function calculates the sum of the array elements and returns the sum in variable *s* in the main program. Notice that the array is called *N* in the main program, but *A* in the function.

3.3.5 *Interrupt Processing*

mikroC Pro for PIC supports interrupts in user programs. When an interrupt occurs, the processor stops whatever it is doing and jumps to the interrupt service routine (ISR). The

www.it-ebooks.info

```
/*-----------------------------------------------------------------------------------------

This program demonstrates hown an array can be passed to a function. Here,
array N is initialized with numbers 1 to 10. Then, function Sum is called to
calculate the sum of the array elements. The sum is then returned to the main
program and stored in variable s


Author:      D. Ibrahim
Date:        October, 2011
File:        Sum.C
-----------------------------------------------------------------------------------------*/
//
// This function calculates the sum of the elements of an array
//
        intSum(int A[ ])
        {
                char i;
                int Total = 0;

                for(i = 0; i < 10; i++) Total = Total + A[i];
                return Total;
        }

//
// Main program
//
void main()
{
        int s, N[10];
        char i;

        for(i = 0; i < 10; i++) N[i] = i + 1;          // Initialize array N
        s = Sum(N);                                    // Calculate the sum of elements
}
```

Figure 3.10 Passing an array to a function

PIC18 series of microcontrollers support both low-priority and high-priority interrupts. High-priority interrupts in mikroC Pro for PIC are handled as user functions, where the reserved word *interrupt* should be used as the function name. An example ISR is shown below:

```
void interrupt(void)
{
        ISR body
}
```

Low-priority interrupts are declared using the keyword *interrupt_low*:

```
void interrupt_low(void)
{
        ISR body
}
```

www.it-ebooks.info

In the PIC18 microcontroller family, the FSR register contents are saved when an interrupt occurs and these registers are restored after returning from the ISR. We can use the following statement to instruct the compiler not to save/restore the register values during an interrupt (except the STATUS, WREG and BSR registers will be saved in high-priority interrupts).

```
#pragma disablecontextsaving
```

Notice that functions can be called from interrupt service routines.

It is also permissible to give any other name to an interrupt service routine, by using the keyword *iv*. The vector address of the ISR must be declared in such cases. An example is given below, which gives the name *MyISR* to a high-priority interrupt:

```
void MyISR() iv 0x00008
{
      ISR body
}
```

3.4 mikroC Pro for PIC Built-in Functions

mikroC Pro for PIC compiler includes a set of built-in library functions that can be used in programs. A list of these functions are given in Table 3.8. There is no need to use header files, except for the functions *Lo*, *Hi*, *Higher* and *Highest*, where the header file *build_in.h* should be included in the program. Built-in functions are implemented as inline, that is the function code is generated at the point of declaration in the program and the function is not called. The only exceptions to this rule are functions *Vdelay_ms*, *Delay_Cyc* and *Get_Fosc_kHz*.

Functions *Lo*, *Hi*, *Higher* and *Highest* are used to extract bytes from integers and long integers, as shown in the following example:

```
a = 0x102578FE;
b = Lo(a);                    //b = 0xFE
c = Hi(a);                    //c = 0x78;
```

Table 3.8 mikroC Pro for PIC built-in functions

Function	Description
Lo	Returns the lowest byte of a number (bits 0 to 7)
Hi	Returns next to the lowest byte of a number (bits 8 to 15)
Higher	Returns next to the highest byte of a number (bits 16 to 23)
Highest	Returns the highest byte of a number (bits 24 to 31)
Delay_us	Creates software delay in microsecond units
Delay_ms	Creates constant software delay in millisecond units
Vdelay_ms	Creates delay in milliseconds using program variables
Delay_Cyc	Creates delay based on microcontroller clock
Clock_Khz	Returns microcontroller clock in KHz
Clock_Mhz	Returns microcontroller clock in MHz

www.it-ebooks.info

```
d = Higher(a);                    //d = 0x25;
e = Highest(a);                   //e = 0x10;
```

Functions *Delay_us* and *Delay_ms* can be used to generate microseconds and seconds delays in programs. For example, to generate a 1 second delay, we can write:

```
Delay_ms(1000);
```

Function *Vdelay_ms* is similar to *Delay_ms*, but here the delay count is a variable, as shown in the following example:

```
MyDelay = 1000;
Vdelay_ms(MyDelay);
```

Function *Delay_Cyc* generates a delay based in the microcontroller clock. The generated delay lasts for 10 times the function parameter in microcontroller cycles. In the following example, the generated delay is 100 microcontroller clock cycles:

```
Delay_Cyc(10);
```

Functions *Clock_kHz* and *Clock_Mhz* return the microcontroller clock frequency in kHz and in Mhz, respectively, nearest to an integer. These functions have no parameters. An example use is shown below:

```
clk = Delay_kHz();
```

3.5 mikroC Pro for PIC Libraries

One of the strong points of mikroC Pro for PIC compiler is that it provides a very rich set of library functions. These functions can be called from user programs without having to use header files. Some of the libraries are hardware specific, such as the USART library, CAN library, PS/2 library, and so on. Some libraries provide ANSI C functions, such as string handling, mathematical functions, and so on. Some libraries provide general purpose functions, such as string conversion, trigonometric functions, time functions, and so on.

Table 3.9 gives a list of the mikroC Pro for PIC libraries. Each library contains a set of functions, and some libraries contain additional libraries. In this section, we shall be looking at the ANSI C library and the Miscellaneous library, which are one of the commonly used libraries. The display libraries (LCD, GLCD, etc.) will be covered in detail in later chapters of this book. Further information can be obtained from the mikroC Pro for PIC manual.

3.5.1 ANSI C Library

The ANSI C library contains the following libraries:

- **Ctype Library:** contains functions for testing characters and converting to upper or lowercase;

www.it-ebooks.info

Table 3.9 mikroC Pro for PIC libraries

Library	Description
ADC	Analogue to digital conversion functions
CAN	CAN Bus functions
CANSPI	SPI based CAN Bus functions
Compact Flash	Compact Flash memory functions
EEPROM	EEPROM memory read/write functions
Ethernet	Ethernet functions
SPI Ethernet	SPI based Ethernet functions
Flash Memory	Flash Memory functions
Graphics LCD	Standard Graphics LCD functions
T6963C Graphics LCD	T6963 based Graphics LCD functions
I^2C	I^2C bus functions
Keypad	Keypad functions
LCD	Standard LCD functions
Manchester Code	Manchester Code functions
Multi Media	Multi Media functions
One Wire	One Wire functions
PS/2	PS/2 functions
PWM	PWM functions
RS-485	RS-485 communication functions
Sound	Sound functions
SPI	SPI bus functions
USART	USART serial communication functions
Util	Utilities functions
SPI Graphics LCD	SPI based Graphics LCD functions
Port Expander	Port expander functions
SPI LCD	SPI based LCD functions
ANSI C Ctype	C Ctype functions
ANSI C Math	C Math functions
ANSI C Stdlib	C Stdlib functions
ANSI C String	C String functions
Conversion	Conversion functions
Trigonometry	Trigonometry functions
Time	Time functions

- **Math Library:** contains functions to perform mathematical operations, such as trigonometric functions sine, cos, tan, logarithms, square-root, and so on;
- **Stdlib Library:** contains functions to convert ASCII characters to integers and vice versa;
- **String Library:** contains functions to append two strings, compare two strings, return the length of a string, and so on.

Example 3.10

Write a function to convert lowercase characters in a string to uppercase, using the mikroC Pro for PIC *Ctype* library function *toupper*.

www.it-ebooks.info

```
/-----------------------------------------------------------------------------------------------------------------

This function converts a string to upper case. The address of the string to be converted
Is passed to the function as a parameter. A while loop is formed to convert each character
To upper case using the toupper library function.


Programmer:   D. Ibrahim
File:         Conv_Upper.C
Date:         October, 2011
-----------------------------------------------------------------------------------------------------------------*/

void Conv_Upper(unsigned char *p)
{
        while(*p != '\0') *p++ = toupper(*p);
}
```

Figure 3.11 Program for Example 3.10

Solution 3.10

The required program listing is shown in Figure 3.11. Function *Conv_Upper* receives the address of the string to be converted into uppercase. A *while* loop is used to convert each character of the string to uppercase using the library function toupper.

3.5.2 Miscellaneous Library

This library contains functions to convert bytes, integers, longs and floating point variables to strings. These functions are used in displays when it is required to display strings or numeric values. An example is given below.

Example 3.11

Use the Miscellaneous library function *ByteToStr* to convert a character to a string.

Solution 3.11

The *ByteToStr* function has two parameters: the byte to be converted, and the address of the string where the converted data will be stored. At least 4 bytes must be reserved for the output string. The required code is given below:

```
char Txt[4];                //Output string
char i = 50;                //Byte to be converted
ByteToStr(i, Txt);          //Txt now contains 50 as a string
```

3.6 Using the mikroC Pro for PIC Compiler

This book is based on the mikroC Pro for PIC compiler, developed by mikroElektronika (www.mikroe.com). A 2 KB size limited demo version of the mikroC Pro for PIC compiler is available from the developers' Web site. This demo version should be sufficient for most

www.it-ebooks.info

small to medium, non-graphical applications. Alternatively, it is recommended that you purchase a full licence of the compiler for developing complex and graphical LCD based applications.

In this section we will be looking at the actual program development, compilation and simulation processes using the mikroC Pro for PIC IDE. After installing the compiler, an icon should appear by default on your Desktop. You should double-click this icon to start the IDE.

3.6.1 mikroC Pro for PIC IDE

After starting the mikroC Pro for PIC, the screen shown in Figure 3.12 will be displayed. The main screen is in two parts. The upper larger part, known as the *Code Editor Window*, is the main screen where programs are written. The lower part is known as the *Message Window*. On the left-hand margin, there are two sliding menus called *Project Settings* and *Code Explorer*. On the right-hand margin, there are four sliding windows called *Library Manager*, *Routine List*, *Project Manager* and *Project Explorer*. Clicking on a sliding window opens the window. Now we briefly look at the functions of various windows.

3.6.1.1 The Code Editor Window

This Code Editor is an advanced text editor and programs are written in this window. The Code Editor supports:

- Code Assistant;
- Parameter Assistant;
- Code Template;
- Bookmarks.

The Code Assistant is useful when writing a program. Type the first few letters of an identifier and press the CTRL+SPACE keys together to obtain a list of all the identifiers beginning with the letters written. For example, type *adc* and press CTRL+SPACE keys. You should see a pop-up window, as shown in Figure 3.13, showing all the identifiers beginning with letters *adc*. Move the cursor up and down to highlight the required identifier and press the ENTER key to make a selection.

The Parameter Assistant is useful when writing programs. When a parenthesis is opened after a function name or an identifier, the Parameter Assistant is invoked automatically to show the type of expected parameters. Figure 3.14 shows an example where the identifier *strlen* is typed followed by a parenthesis. Here, the expected type is shown to be *unsigned char ∗s*.

The Code Template is used to generate code templates in our programs. Using this feature can save us typing long keystrokes. An example is given in Figure 3.15. Here, assume that we wish to create a loop using the *do – while* statements. The identifier *do* is typed first, followed by pressing CTRL+J keys simultaneously. A pop-up window shows any options available. By highlighting the required option and pressing the ENTER key, the template for the *do – while* repetition statement is created.

www.it-ebooks.info

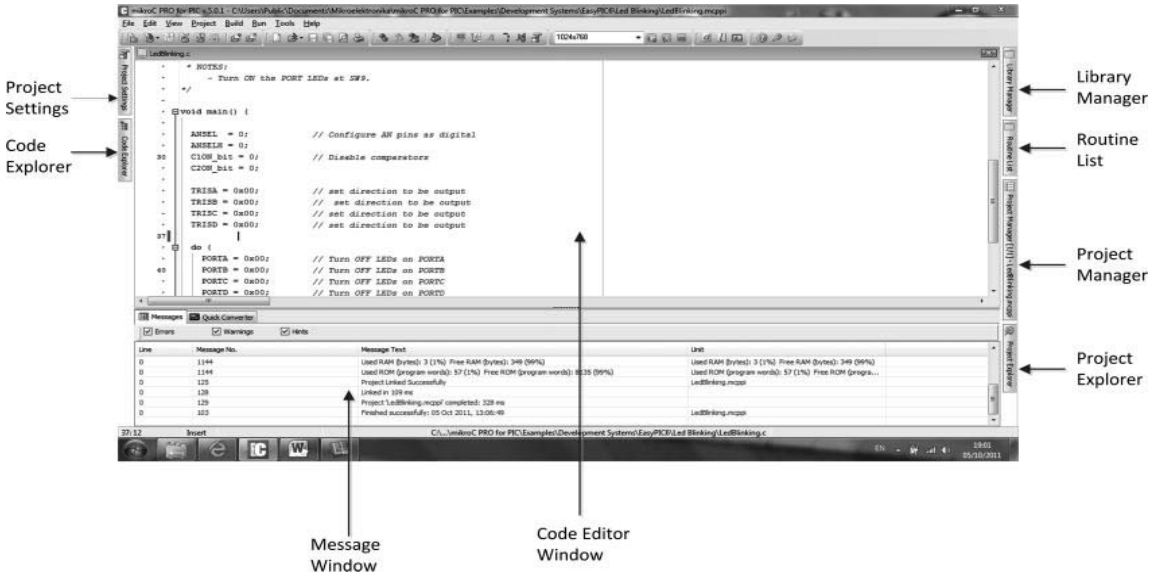


Figure 3.12 mikroC Pro for PIC IDE screen

www.it-ebooks.info

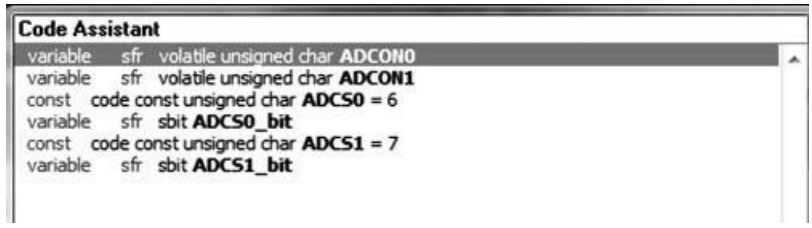

Code Assistant
variable sfr volatile unsigned char ADCON0
variable sfr volatile unsigned char ADCON1
const code const unsigned char ADCS0 = 6
variable sfr sbit ADCS0_bit
const code const unsigned char ADCS1 = 7
variable sfr sbit ADCS1_bit

Figure 3.13 The Code Assistant

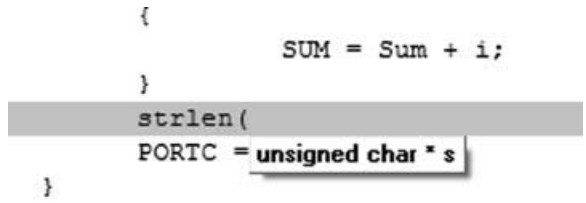


Figure 3.14 The Parameter Assistant

Bookmarks make navigation through a large code easier. To set a bookmark, use the key combination CTRL+Shift+Number. The same principle applies to the removal of the bookmarks. To jump to a bookmark, use the key combination CTRL+Number. For example, in Figure 3.16, a book mark with number 5 is placed on line 36 of the code by typing CTRL+Shift+5. Notice that the bookmark line is identified by the bookmark number, followed by an arrow in the left-hand margin. Then, by typing CTRL+5 from anywhere in the program, the cursor will jump to the point where bookmark 5 was placed.

Figure 3.15 The Code Template

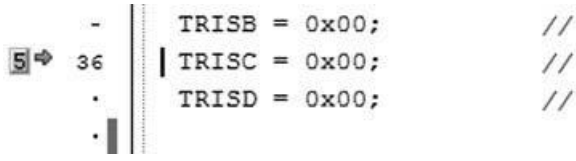


Figure 3.16 Showing bookmark 5

www.it-ebooks.info

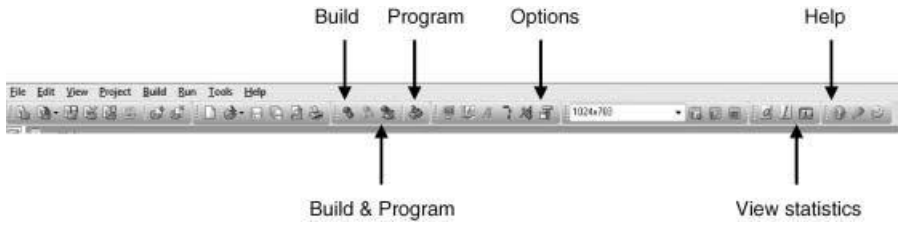


Figure 3.17 Commonly used toolbars

The Code Editor window has a menu at the top and a large number of toolbars for selecting various options. Figure 3.17 shows some of the most commonly used toolbars of this window.

3.6.1.2 The Message Window

The Message window consists of two tabs: the *Messages* and *Quick Converter*. When the Messages tab is selected (see Figure 3.18), the window displays information about the current compilation, such as the compile time, amount of program and data memory used, and any error messages. The Quick Converter window is useful for converting numbers from different bases, and for floating point number conversions. In Figure 3.19, an example is shown, where decimal number 65 is converted into various bases.

3.6.1.3 The Project Settings Window

With the Project Settings window you can define the microcontroller type, the clock frequency, and the build/debugger type to be used in the project. Figure 3.20 shows an example where the device type is PIC16F887, the clock frequency is 8.0 MHz, the build type is Release, and the debugger type is Software.

3.6.1.4 The Code Explorer Window

The Code Explorer window gives a clear view of each item declared inside the source code.

You can jump to a declaration of any item by right clicking it. Also, besides the list of defined and declared objects, code explorer displays message about first error and its location in code. Figure 3.21 shows an example code explorer window.

3.6.1.5 The Library Manager Window

The Library Manager window enables the user to add or remove libraries from a project. Figure 3.22 shows part of the library manager window.

3.6.1.6 The Routine List Window

The Routine List window displays a list of all the routines (functions) used in a program, with the line numbers where they are used. In Figure 3.23, only function *main* is used in the

www.it-ebooks.info

Messages
Quick Converter
Errors
Warnings
Hints
Line
Message No.
Message Text
Unit
0 1144 Used RAM (bytes): 3 (1%) Free RAM (bytes): 349 (99%) Used RAM (bytes): 3 (1%) Free RAM (bytes): 349 (99%)
0 1144 Used ROM (program words): 57 (1%) Free ROM (program words): 8135 (99%) Used ROM (program words): 57 (1%) Free ROM (progra...
0 125 Project Linked Successfully LedBlinking.mcppi
0 128 Linked in 63 ms
0 129 Project LedBlinking.mcppi' completed: 266 ms
0 103 Finished successfully: 05 Oct 2011, 19:45:19 LedBlinking.mcppi

Figure 3.18 The Message window

www.it-ebooks.info

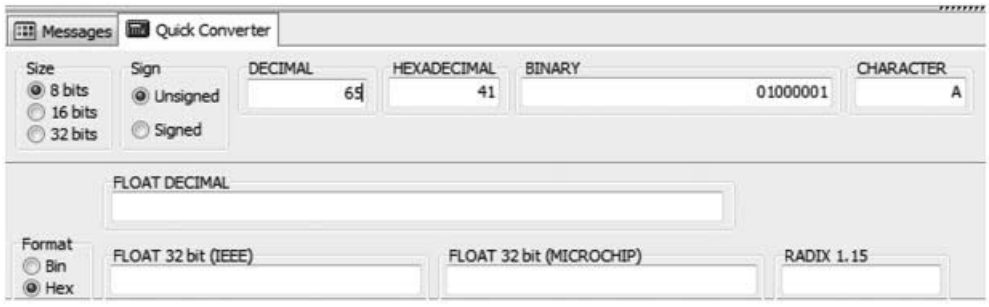


Figure 3.19 The Quick Converter window

program and it appears at line 26 of the program. The list can be sorted either in line number order, or in name order.

3.6.1.7 The Project Manager Window

The Project Manager window allows users to manage multiple projects. In general, several projects, which together make a large project group, may be open at the same time. Only one of them may be active at any one moment in time. Setting a project in active mode is

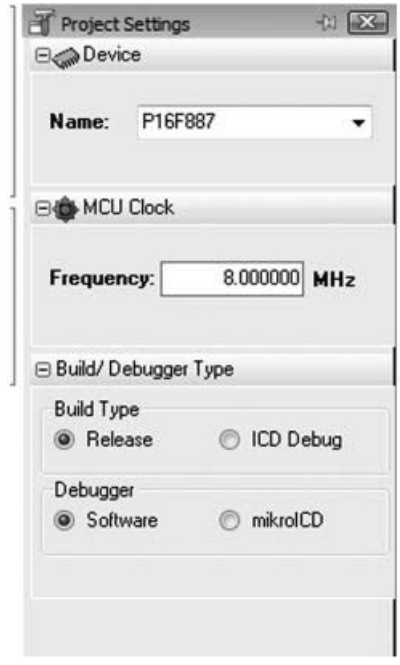


Figure 3.20 Project settings window

www.it-ebooks.info

Figure 3.21 Code Explorer window

performed by double clicking on the desired project in the Project Manager window. An example project manager window is shown in Figure 3.24.

3.6.1.8 The Project Explorer Window

The Project Explorer window is a collection of program examples for various development boards, PIC microcontroller chips, and various interesting routines. An example window is shown in Figure 3.25.

3.6.2 Creating a New Source File

A mikroC Pro for PIC project may consist of several files, such as source files, listing files, hex files, assembler files, configuration files, and so on. All the files related to a project are stored in a project file having the extension '.mcppi'. C source files created by the user have extensions '.c'. In this section we shall be looking at an example and learn step by step how to create a new source file.

Example 3.12

Write a C program to calculate and display the squares of numbers from 1 to 10 on PORT C of a microcontroller. Assume that a PIC16F887 type microcontroller is to be used with 8.0 MHz clock, and the filename is *sum.c*.

Solution 3.12
The step-by-step solution is given below:

Step 1: Start mikroC Pro for PIC IDE by double clicking on the icon in Desktop.

www.it-ebooks.info

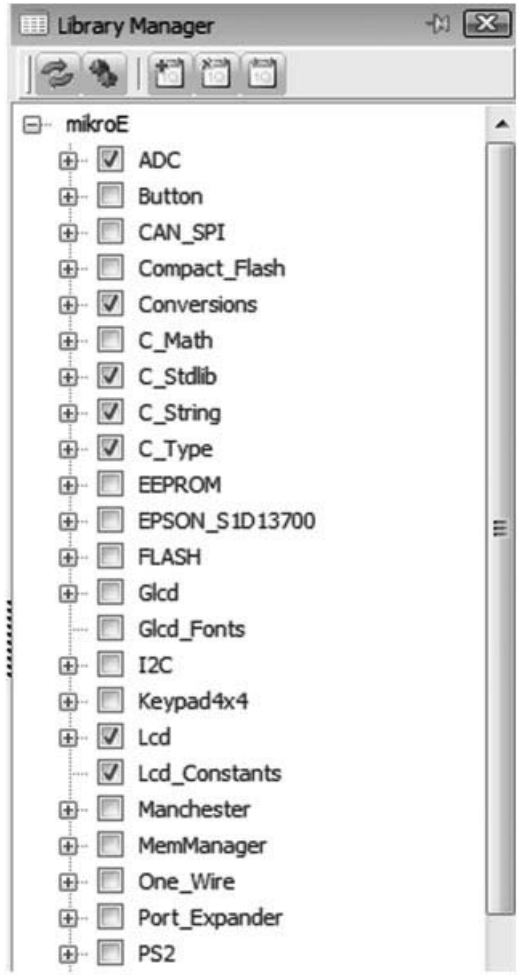


Figure 3.22 Library Manager window

Figure 3.23 Routine List window

www.it-ebooks.info

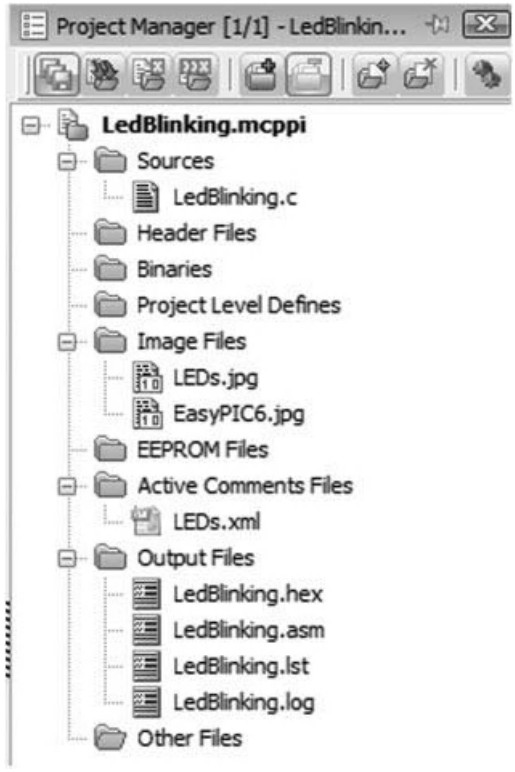


Figure 3.24 Project Manager window

Step 2: Create a new project by clicking *Project -> New Project-> Next* and enter a project name (here *SUM* is chosen), choose a project folder (here folder *Alevs* is chosen), select device type as PIC16F887 and the clock rate 8.0 MHz. Figure 3.26 shows the new project window.

Step 3: Click *Next* and *Next* again as the source file is not ready and we will be writing it using the mikroC Pro for PIC editor. Select all libraries for the project, as shown in Figure 3.27.

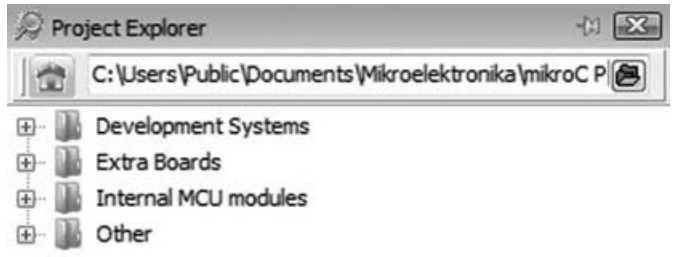


Figure 3.25 Project Explorer window

www.it-ebooks.info

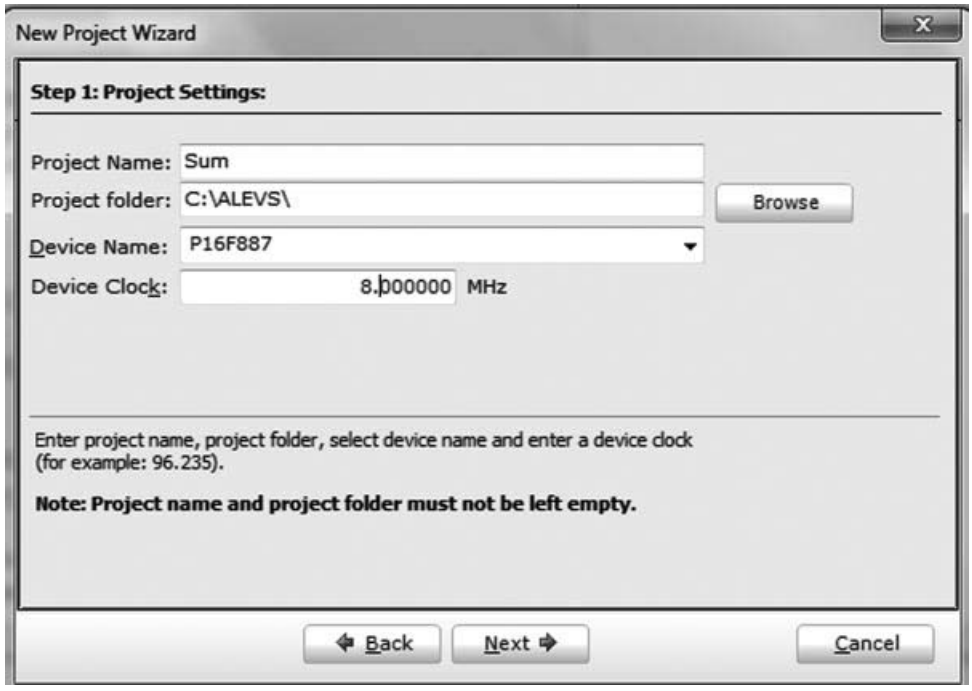


Figure 3.26 New project window

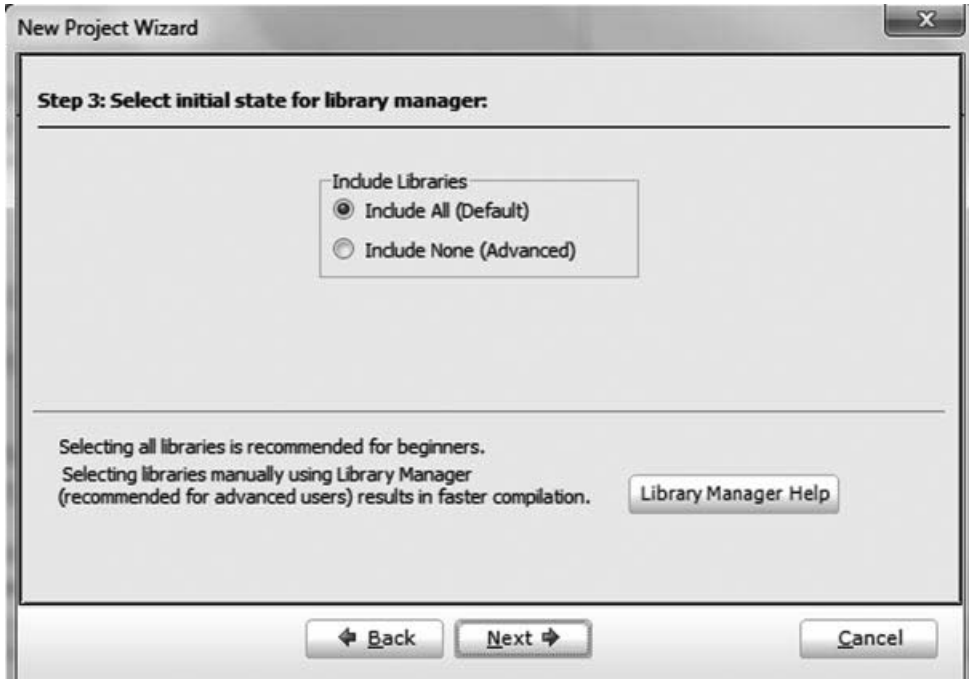


Figure 3.27 Selecting all files

www.it-ebooks.info

Step 4: Click *Next* and *Finish* to complete the new project creating steps. You should now see a blank IDE. Enter the following program into the Code Editor section of the IDE:

```
/*-------------------------------------------------------
                    Squares of Numbers 1 to 10
                    ----------------------
This program calculates and displays the squares of integer numbers
from
1 to 10 on PORT C of a PIC16F887 microcontroller.

Programmer: D. Ibrahim
File:       Sum.C
Date:       October, 2011
-------------------------------------------------------*/
void main()
{
      unsigned char i, Squares;
      TRISC = 0;                              //Configure PORT C as output

      for(i = 1; i <=10; i++)
      {
                Squares = i*i;                //Calculate the square
      }
      PORTC = Squares;                        //Send result to PORT C
}
```

PIC microcontroller input-output port pins are bi-directional. Notice in this example that PORT C input-output pins are configured as output by clearing the TRISC register. A zero in a TRIS register bit position forces the corresponding PORT pin to be an output. Similarly, a 1 in a TRIS register bit position forces the corresponding PORT pin to be an input.

Figure 3.28 shows the program in the Code Editor window.

3.6.3 Compiling the Source File

To compile the program, either click the Build toolbar shown in Figure 3.17, or press keys CTRL+F9 simultaneously. Any compilation errors will be displayed in the Message window at the bottom part of the screen. If there are no errors then the message, *Finished successfully*, will be displayed, followed by the current date and time.

After a successful compilation, the compiler generates the *Hex* file that can be downloaded to the target microcontroller using a programmer device. Many companies offer microcontroller development kits with built-in programmers. For example, the *easyPIC6* is a PIC microcontroller development board with built-in programmer and debugger. This kit is fully compatible with the mikroC Pro for PIC compiler. Programs compiled using the mikroC Pro for PIC compiler can easily be loaded to the target microcontroller on the *easyPIC6* development board.

The Hex file for this example is shown in Figure 3.29.

www.it-ebooks.info

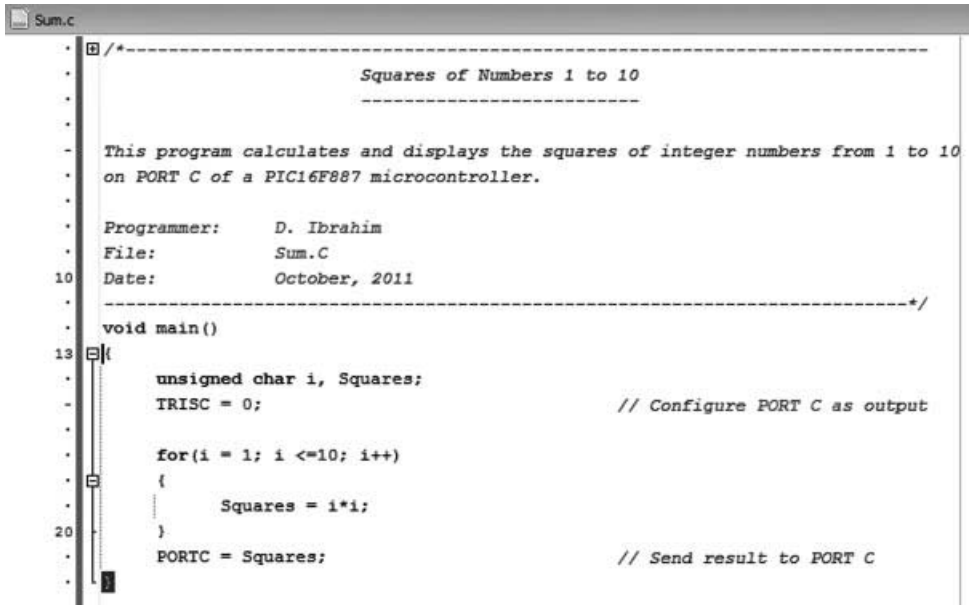


Figure 3.28 Program in the Code Editor window

```
:020000002F28A7
:0E00060083120313210 88A00200882000800DC
:10001400831203137008F100F0010830FC0071082A
:10002400F40C03181928FC0B1228F10100340310F6
:100034001E28F40C0318F107F10CF00CFC0B1B2820
:020044000800B2
:1000460003208A110A128000840AA00A0319A10A51
:08005600F003031D232808003C
:10005E00831603138701013083 12A20022080A3C83
:10006E00031C42282208F0002208F4000A2070081F
:0C007E00A300A20A3528230887004428AC
:04400E00F72F000781
:00000001FF
```

Figure 3.29 The generated Hex file

3.7 Using the mikroC Pro for PIC Simulator

mikroC Pro for PIC IDE includes a powerful and easy-to-use simulator that can be extremely useful during program development and testing. The simulator is a program and does not need any hardware for its operation. Using the simulator we can step through a program, observe and, if required, change the values of variables as the program is running, insert break-points in the program and run the program until the break-point is hit, and so on.

An example use of the simulator is given below, step by step, by considering the program developed in Example 3.12.

www.it-ebooks.info

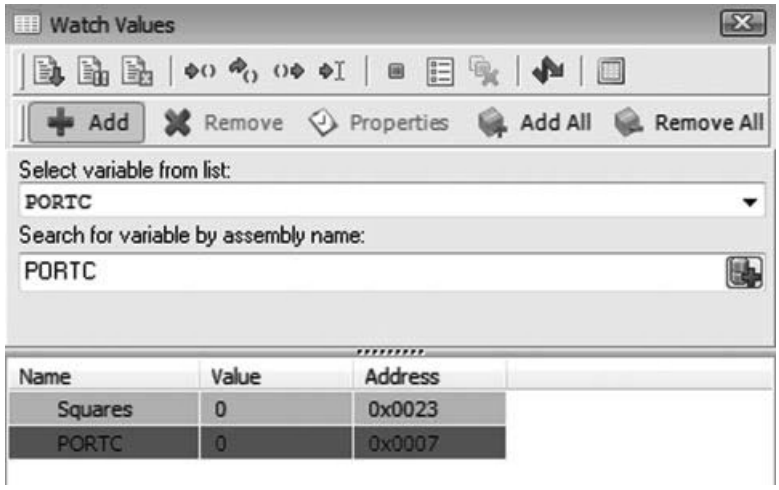


Figure 3.30 The simulator window

Step 1: Compile the program by making sure that the *Build Type* is set to *Release* and the *Debugger* is set to *Software* in the *Project Settings* window.

Step 2: From the drop-down menu select *Run -> Start Debugger*, or press the F9 function key. You will now see the simulator window on the right-hand side. A blue bar across the screen on the left-hand side points to the first executable statement in our program.

Step 3: Select the variables to be monitored using the simulator window. In this example, we wish to monitor the values of variable *Squares* and *PORTC*.

Click to open the list box *Select variable from list*: under *Watch Values* in the simulator window. Select variable *Squares* from the list box. Then click *Add* under *Watch Values* to add this variable to the monitor list. Repeat for the PORTC. You should now have the simulator window, as in Figure 3.30.

Step 4: We are now ready to single-step our program and observe values of the selected variables as the program is running. Press function key F8 to single-step through the program. The blue bar should move to the *for* statement. Pressing F8 again should execute this statement and the blue bar should move to statement where the square of the number is calculated. Pressing F8 again will execute this statement. The value of variable *Squares* will change to 1 in the Simulation window. Keep pressing F8 and you will see the value of *Squares* changing. The value of any variable can also be displayed by moving the cursor over the variable. For example, move the cursor over variable *i* to see its value at any instant in time. After executing the loop 10 times you should see that the final value of *Squares* is 100 and PORT C is also set to this value. Figure 3.31 shows the Simulator window at the end of the simulation.

3.7.1 Setting a Break-Point

Break-points are useful when we want to execute the program up to a point and then display (or change) the variables at this point. In this example we will set a break-point at the statement, which sends the value of *Squares* to PORTC. The steps are given below:

www.it-ebooks.info

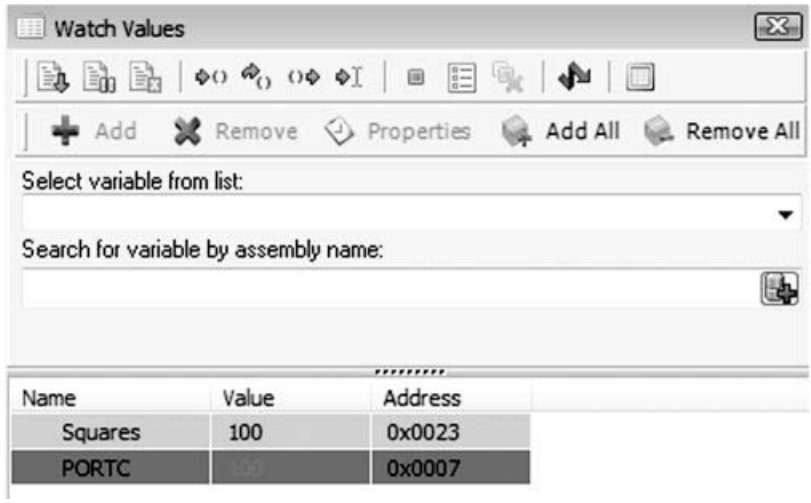


Figure 3.31 End of the simulation

Step 1: Start the simulator and select the variables to be monitored, as described in steps 1–3 before.

Step 2: Place the cursor on the line where a break-point is to be placed and select *Run -> Toggle Breakpoint* from the drop-down menu (or press the F5 function key). A red bar will appear on the line where the break-point is placed, and a red dot will be displayed on the line at the left-hand margin, as shown in Figure 3.32.

Step 3: *Select Run -> Run/Pause debugger* (or press function key F6) to run the program until the break-point is hit. At this point you should see that variable *Squares* has value

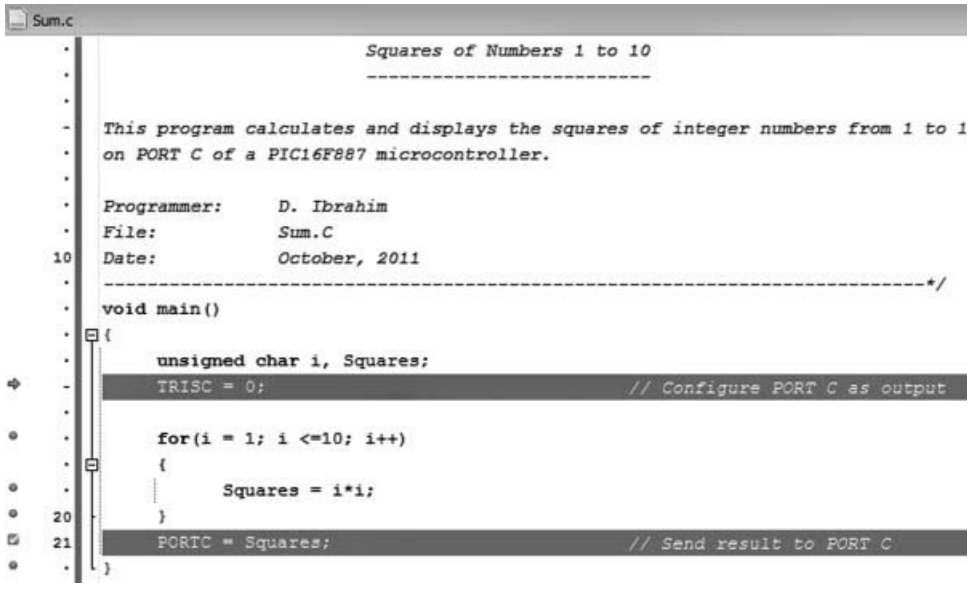

Sum.c
Squares of Numbers 1 to 10
This program calculates and displays the squares of integer numbers from 1 to 1
on PORT C of a PIC16F887 microcontroller.
Programmer: D. Ibrahim
File: Sum.C
Date: October, 2011
void main()
{
unsigned char i, Squares;
TRISC = 0; // Configure PORT C as output
for(i = 1; i <=10; i++)
{
Squares = i*i;
}
PORTC = Squares; // Send result to PORT C
}

Figure 3.32 Setting a break-point

www.it-ebooks.info

100 and PORTC has value 0. Press F8 to execute the next statement, which will set PORTC to 100.

To remove all the break-points, select *Run -> Clear Breakpoints* (or press SHIFT+CTRL+F5) from the drop-down menu. To remove a specific break-point, click on the red dot at the left-hand line margin where the break-point is placed.

3.8 Other mikroC Pro for PIC Features

mikroC Pro for PIC IDE includes a number of useful features for programmers. Some of these features are described briefly in this section.

3.8.1 View Statistics

This window shows the variables, memory usage and function usage in our program. Select *View -> Statistics* from the drop-down menu (or click the toolbar as shown in Figure 3.17). Figure 3.33 shows the memory usage statistics in graphical form for the Example 3.11. In this example, 98.6% of the data memory and 99.2% of the program memory are free.

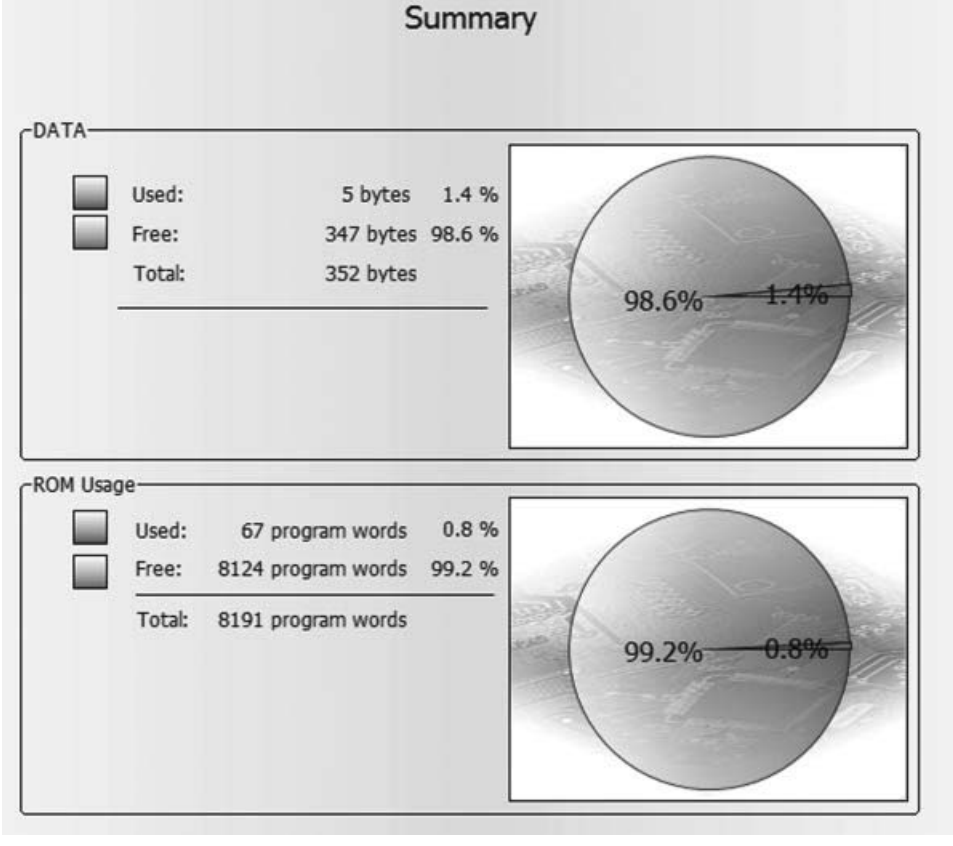


Figure 3.33 Memory usage statistics

www.it-ebooks.info

```
1
.   _main:
.
.   ;Sum.c,12 ::                    void main()
-   ;Sum.c,15 ::                    TRISC = 0;
.           CLRF        TRISC+0
.   ;Sum.c,17 ::                    for(i = 1; i <=10; i++)
.           MOVLW       1
.           MOVWF       main_i_L0+0
10  L_main0:
.           MOVF        main_i_L0+0, 0
.           SUBLW       10
.           BTFSS       STATUS+0, 0
.           GOTO        L_main1
-   ;Sum.c,19 ::                    Squares = i*i;
.           MOVF        main_i_L0+0, 0
.           MOVWF       R0+0
.           MOVF        main_i_L0+0, 0
.           MOVWF       R4+0
20          CALL        _Mul_8x8_U+0
.           MOVF        R0+0, 0
.           MOVWF       main_Squares_L0+0
```

Figure 3.34 Assembly listing of Example 3.12

3.8.2 *View Assembly*

This window is selected by *View -> Assembly* from the drop-down menu and it shows the Assembly listing of our program. Figure 3.34 shows part of the Assembly listing of Example 3.12.

3.8.3 *ASCII Chart*

The ASCII chart can be displayed by clicking toolbar ASCII Chart.

3.8.4 *USART Terminal*

A USART terminal is available by clicking toolbar USART Terminal. This terminal is useful when developing serial RS232 based projects.

3.8.5 *Seven Segment Editor*

The 7-segment editor is displayed by clicking the 7 sign on the toolbars. The editor is useful when we need to create fonts using the 7-segment displays. An example is shown in Figure 3.35.

www.it-ebooks.info

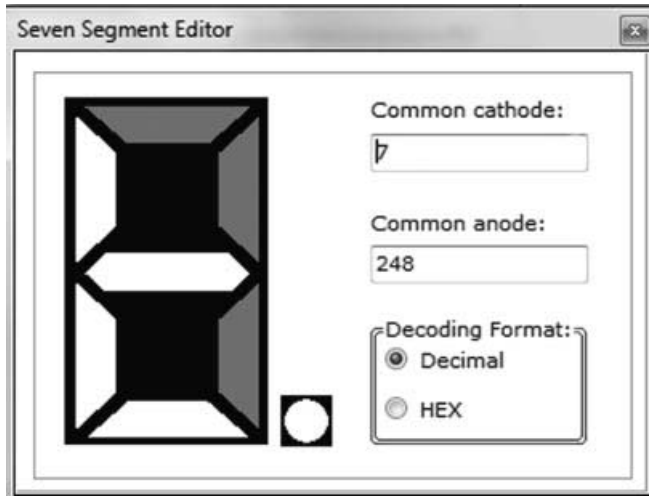


Figure 3.35 The Seven Segment Editor

3.8.6 Help

The help window could be useful when we are looking for help on compiler specific issues. This window is selected from the toolbars (see Figure 3.17).

3.9 Summary

mikroC Pro for PIC is one of the most commonly used high-level languages for programming PIC microcontrollers. This chapter presented a brief introduction to the mikroC Pro for PIC language.

Variables in a mikroC Pro for PIC language store values during running of a program and they can be 8-bit, 16-bit, 32-bit or floating point. Variables are stored in the data memory of the target microcontroller. Constants, on the other hand, are variables that have fixed values and they are stored in the flash memory (program memory) of the target microcontroller.

mikroC Pro for PIC supports several statements for changing the flow of control in a program, such as if, else, do, while, switch and break statements.

Pointers are useful features of the C language and they are used to store the memory addresses of variables. By using pointers, we can easily pass data to functions and manipulate arrays.

The mikroC Pro for PIC IDE is a powerful and easy-to-use environment for creating, editing and compiling our programs. In addition, the IDE supports simulation and debugging features, which helps in the development and testing of programs. The use of the simulator has been described in this chapter, step by step, in a tutorial fashion.

The mikroC Pro for PIC IDE supports some other useful features, such as memory, function and variable usage statistics, Assembly listing of the created program, 7-segment display font creation, RS232 USART terminal, ASCII listing, EEPROM editor, online Help, and so on.

www.it-ebooks.info

Exercises

3.1 What does program repetition mean? Describe the operation of *for*, *while* and *do – while* loops in C.

3.2 Write a program to calculate the sum of numbers from 1 to 10 and send the result to PORT C of a PIC microcontroller.

3.3 Write a function to calculate the squares of numbers between 1 and 10 and store the results in an array passed as a function parameter.

3.4 What can you say about the following *while* loop?

```
i = 10;
while(i < 100)
{
        Sum++;
        j = Sum;
}
```

3.5 Write a program to calculate the sum of even numbers between 0 and 100.

3.6 Write a function to calculate the volume of a cube. Show how this function can be called from a main program.

3.7 Write a function to convert inches to centimetres. Show how this function can be called from a main program to convert 12 inches to centimetres.

3.8 Write a function to add two vectors of dimensions 15, passed as parameters to the function.

3.9 Write two functions to convert temperature from Fahrenheit to Celsius and vice-versa. Show how the two functions can be combined.

3.10 Find the errors in the following program:

```
void main()
{
    chr i;
    integer j;

    for(i = 0; i < 10; i+)
    {
        j++;
    }
    Total = j;
```

3.11 What is an array? How are arrays defined? Write example statements to define an array of 50 integers.

3.12 Using a *for* loop, write a program to initialise the elements of an array to 1. Assume that the array dimension is 30.

3.13 How is the text string 'Computer' represented in C? Describe different ways of declaring this string in a program.

3.14 Write a function to receive a string and then to convert first the character of this string to uppercase.

www.it-ebooks.info

3.15 Write a function to check whether or not the first character of a string is uppercase. Return 1 if it is upper case, otherwise return 0.

3.16 Write a function to calculate the number of spaces in a character string and return this number as an integer.

3.17 Write a function to join two strings whose addresses are supplied as function parameters.

3.18 What is wrong with the following code?

```
char i, j;
for(i = 0; i < 2500; i++) j++;
```

3.19 Re-write the following expressions in a different way, to remove the logical NOT:

```
a) !(b > 3)        b) !((a + b) == c)     c) !(x == 30)
```

3.20 Write the equivalent *if-else* statements for the following tests:

```
a) (a > b) ? 2: 1    b) (a == 10) ? 0:1    c) ((a + b) > 2) ? 1:0
```

3.21 What is a *void* function? Describe with an example.

3.22 What is a function prototype? Describe with an example.

3.23 How would you use the *sizeof* operator to calculate the storage requirements of a structure? Give an example.

3.24 What will be the value of variable *Total* in the following example?

```
Total = 100;
for(i = 0; i <10; i++)Total--;
```

3.25 What will be the value of variables *x* and *y* after the following expressions are executed?

```
x = 12;
a) y = --x        b) y = x++        c) y += x
```

3.26 Given that ptr is a pointer to an integer variable, what are the results of the following expressions?

```
a) *ptr        b) *(ptr + 1)        c) ptr + 2
```

www.it-ebooks.info

4

PIC Microcontroller Development Tools – Including Display Development Tools

The development of a microcontroller based system used to be a difficult task, but nowadays this task is simplified considerably with the availability of many development tools. In general, the development tools are hardware and software based, although much of the tools are nowadays integrated, enabling the user to easily create a program, test it, and load it to the target hardware with the click of a button.

The tools for creating a microcontroller based project include visual text editors, assemblers, compilers, simulators, in-circuit debuggers, emulators, device programmers and development kits.

A typical project development cycle starts with designing the hardware. Then the program is written using a text editor. The Windows operating system is distributed with a text editor called the Notepad. Note that you cannot use the Word program as a text editor, as the written text is embedded with special control characters. The assemblers or compilers do actually contain built-in text editors that can be used for program development. The developed program is then converted into a form that can be understood by the microcontroller using an assembler or a compiler. Some large programs consist of several modules and such modules are combined using a linker program. The final program is then usually simulated in software. Simulation is an invaluable tool, as it helps the programmer to detect any errors in early stages of the software development. Using the simulator program, the programmer can step through the code and observe the values of variables, or change them as required and make sure that the program is actually doing what it is supposed to do. Once the programmer is happy with the developed code, the code can be loaded into the program memory of the target microcontroller using either device programmer hardware or a development kit. The program is then tested on the actual hardware using various hardware tools such as an in-circuit debugger or an emulator. With the help of these tools, the programmer can observe

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

the values of variables and registers as the program is running either in single step mode or in real-time run mode.

Nowadays, most companies offer integrated hardware and software packages where tools such as editors, assemblers, compilers and simulators are collected under a single software development tool. The in-circuit debuggers and device programmers are then available on a development kit. Users can develop their programs, and then compile and simulate the program with the help of the software development tools. Once the program is working, the executable code is transferred to the target microcontroller chip with the help of a device programmer or usually a hardware development board. This process is usually done with the click of a button. The development kit contains the target microcontroller, and in addition, various peripheral devices to help test the program, such as LEDs, LCDs, push-button switches, power supply, in-circuit debugger, and so on.

In this chapter we will be briefly looking at some of the commonly used PIC microcontroller hardware development boards. In addition, some of the popular display development boards will be investigated.

4.1 PIC Hardware Development Boards

Microcontroller development boards are available in many shapes and sizes. Some simple boards include a microcontroller, clock circuitry, power supply and a few LEDs. Some more advanced boards contain LCDs, graphics LCDs (GLCDs), push-button switches, USB ports, CAN bus ports, serial communication ports, in-circuit-debugging module, and so on.

Some of the commonly used development boards and their specifications are described in this section.

4.1.1 Super Bundle Development Kit

The Super Bundle development kit (see Figure 4.1), manufactured by microEngineering Labs Inc., is a complete integrated development board with the following features:

- PICBASIC PRO compiler;
- MicroCode Studio Plus integrated development environment with in-circuit debugger;
- LAB-X1 experimenter board with 5 V power supply, 40-pin ZIF socket, oscillator, reset circuit, LEDs, LCD, serial EEPROM, real-time clock, temperature sensor, servo drive, RS232/RS485 ports, IR interface, speaker and prototyping area;
- microcontroller programmer and programming adaptor;
- mains adaptor, USB cable and serial cable.

4.1.2 PIC18 Explorer Board

The PIC18 Explorer Board (see Figure 4.2) manufactured by Microchip Inc., can be used in the development of PIC18 microcontroller based projects. The board contains:

- sample PIC18F8722 and PIC18F87J11 (plug-in module) microcontrollers;
- 28–80 pin microcontroller support;

www.it-ebooks.info

Figure 4.1 Super Bundle development kit. (Reproduced with permission from microEngineering Labs)

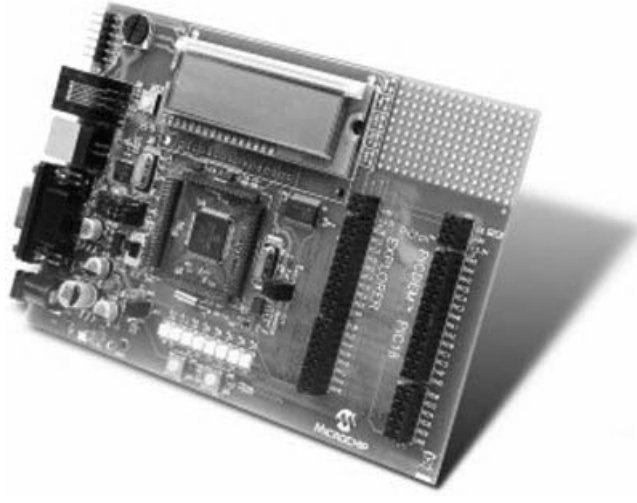


Figure 4.2 PIC18 Explorer board. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

- LEDs and LCD display;
- expansion board connector (PICtail), enabling a large number of external boards to be connected;
- crystal oscillator;
- potentiometer (connected to A/D converter);
- USB and RS232 interface;
- programmable power supply;
- temperature sensor;
- MPLAB, ICD3 and REAL ICE connectors;
- SPI EEPROM;
- prototyping area.

4.1.3 PIC18F4XK20 Starter Kit

This board (see Figure 4.3) is manufactured by Microchip Inc., and can be used as a demonstration and learning board. The board contains

- PICkit 2 programmer/debugger;
- 128/64 Organic LED display;
- 32.768 kHz external clock;
- 4 push-button switches;
- 8 LEDs Serial EEPROM;

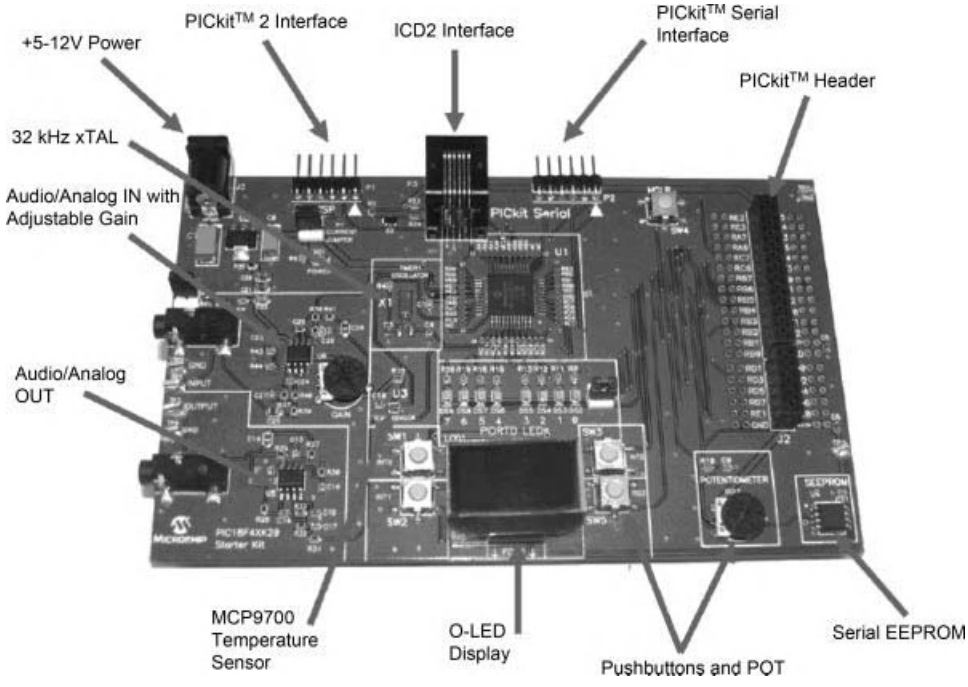


Figure 4.3 PIC18F4XK20 Starter Kit. (Reproduced with permission from Microchip Inc)

www.it-ebooks.info

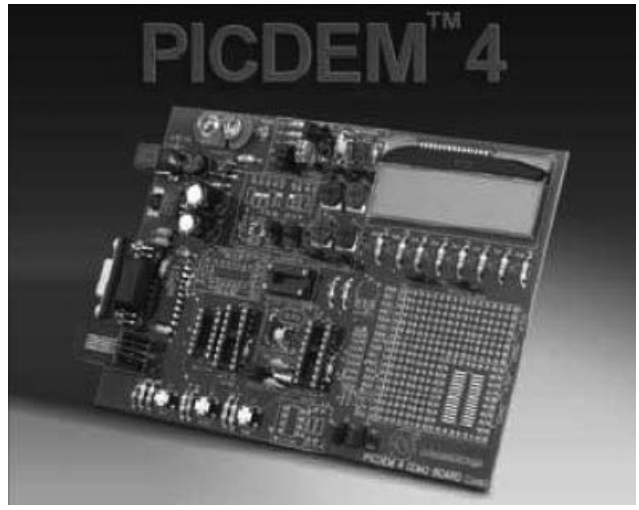


Figure 4.4 PICDEM 4 development board. (Reproduced with permission from Microchip Inc)

- daughter board (PICtail) for connecting external boards;
- potentiometer;
- Analogue in/out;
- ICD2 interface;
- temperature sensor.

4.1.4 PICDEM 4

The PICDEM 4 board (see Figure 4.4), manufactured by Microchip Inc., can be used for developing 8-, 14- and 18-pin microcontroller based projects. The board contains:

- supporting 8-, 14- and 18-pin DIP devices;
- on-board +5 V regulator for direct input from AC/DC wall adaptor;
- RS-232 port;
- 8 LEDs;
- 2 × 16 LCD display;
- 3 push button switches and master reset;
- prototyping area;
- I/O Expander;
- supercapacitor circuitry;
- area for a LIN transceiver;
- area for a motor driver;
- MPLAB ICD 2 connector.

4.1.5 PIC16F887 Development Kit

This kit (see Figure 4.5) is manufactured by Custom Computer Services Inc. The kit includes an optional C compiler. The main features of this kit are:

www.it-ebooks.info

Figure 4.5 PIC16F887 development kit. (Reproduced with permission from Custom Computer Services Inc)

- PIC16F887 prototyping board (see Figure 4.6);
- 30 I/O pins;
- in-circuit debugger/programmer;
- potentiometer;
- push-button switch;

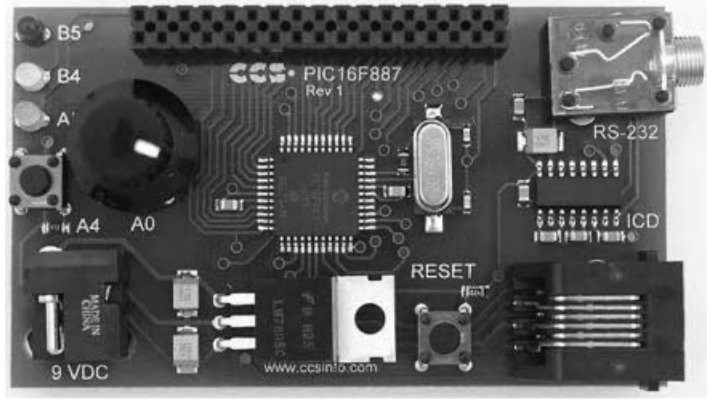


Figure 4.6 PIC16F887 prototyping board

www.it-ebooks.info

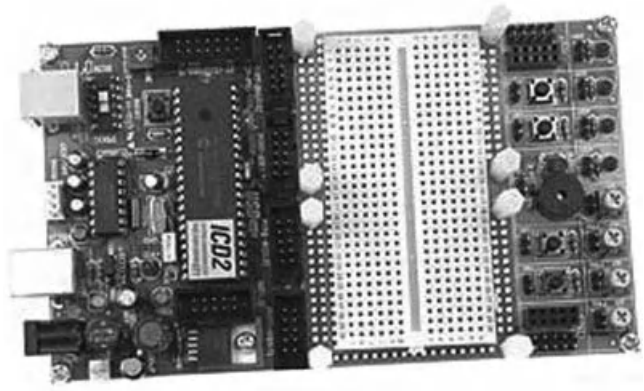


Figure 4.7 FUTURLEC PIC18F4550 development board. (Reproduced with permission from Futurlec Inc)

- RS232 converter;
- breadboard;
- serial EEPROM;
- real-time clock chip;
- digital temperature chip;
- two-digit 7-segment display;
- power supply.

4.1.6 FUTURLEC PIC18F4550 Development Board

Although this board (see Figure 4.7) has been developed for USB based applications, it can be used in general microcontroller based project development. The board has the following features:

- includes PIC18F4550 Microcontroller;
- all necessary power supply components are pre-installed and board is ready-to-run;
- direct In-Circuit Program Download with the PIC Programmer or Microchip ICD2;
- 4 pushbuttons with Speaker;
- 4 variable potentiometers;
- 4 LEDs;
- large Breadboard area;
- USB and RS232 Connection;
- selectable PROG-RUN Switch;
- power and Programming LED;
- In-Circuit Debugging with Microchip ICD2 Unit.

4.1.7 EasyPIC6 Development Board

The EasyPIC6 development board (see Figure 4.8) is a sophisticated development board, manufactured by mikroElektronika. The board is fully integrated with the compilers

www.it-ebooks.info

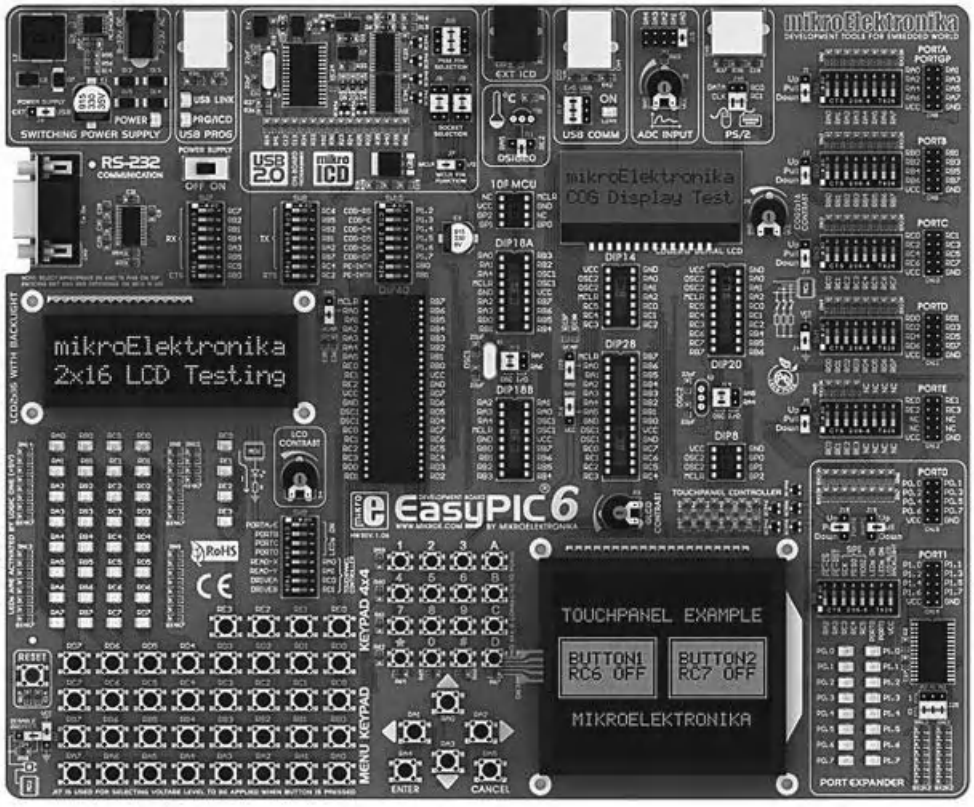


Figure 4.8 EasyPIC6 development board. (Reproduced with permission from mikroElectronika)

developed by the company. Thus, a developed program can very easily be loaded to the program memory of the target microcontroller. In addition, the board contains an in-circuit debugger that can be used during the project development.

The main features of the EasyPIC6 development board are:

- support for over 160 PIC microcontrollers from 8- to 40-pin;
- 2 × 16 LCD and 2 × 16 COG (Chip-On-Glass) LCD;
- 128 × 64 GLCD;
- In-circuit debugger (mikroICD) and programmer;
- 4 × 4 keypad;
- USB communication port;
- 36 push-button switches;
- 36 LEDs External or USB power supply;
- digital thermometer;
- on board crystal oscillator;
- reset button;
- menu keypad;

www.it-ebooks.info

- port expander circuit;
- 2 potentiometers for the analogue inputs;
- RS232 and PS/2 ports;
- all port pins available at IDC10 connectors.

4.1.8 EasyPIC7 Development Board

This is the latest development board (see Figure 4.9) from mikroElektronika, offering a large number of interface devices, and supporting over 250 PIC microcontroller types. The board is fully compatible and fully integrated with all the PIC microcontroller based compilers (mikroC Pro for PIC, mikroBASIC Pro for PIC, and mikroPASCAL Pro for PIC) offered by the company.

The main features of the EasyPIC7 development board are:

- support for over 250 PIC microcontrollers from 8- to 40-pin;
- 2 × 16 LCD;
- 128 × 64 GLCD with touch panel circuit;
- in-circuit debugger (mikroICD) and programmer;
- buzzer;
- digital (DS1820) and analogue (LM35) Temperature sensors;
- USB communication port;
- 36 push-button switches;
- 36 LEDs EEPROM;
- mikroBUS for connecting external boards;
- external or USB power supply;
- on board crystal oscillator;

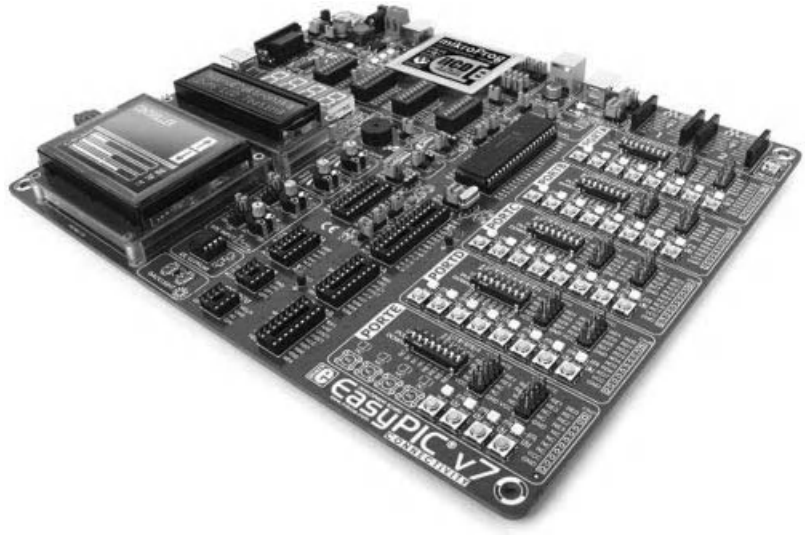


Figure 4.9 EasyPIC7 development board

www.it-ebooks.info

- ICD2/ICD3 connector;
- external and USB dual power supply (5 V and 3.3 V);
- reset button;
- 4-digit 7-segment display;
- 2 potentiometers for the analogue inputs;
- tri-state DIP switches;
- RS232;
- all port pins available at IDC10 connectors.

4.2 PIC Microcontroller Display Development Tools

Display development tools are both hardware and software tools used during the development of display based projects. In this section we will look at both types of tools briefly. In the following chapters, the use of these tools will be described in detail, together with actual physical projects.

4.2.1 Display Hardware Tools

Some of the displays, such as LEDs, 7-segment displays and text based LCDs are simply connected directly to the I/O ports of microcontrollers. These simple display devices are driven directly by software written by the user. In this section we are more concerned with GLCD display devices with integrated microcontrollers. These are in general small handheld devices incorporating GLCD displays on one side of the PCB, and microcontroller hardware on the other side of the PCB. The microcontroller is programmed for a specific display task, and once programmed the devices can be used stand-alone, for example by connecting to a battery. Some displays incorporate Touch Screen panels and circuits to enable the user to make selections by touching the screen.

4.2.1.1 SmartGLCD Board

SmartGLCD board (see Figure 4.10) is a small (14×9 cm) handheld GLCD display development board manufactured by mikroElektronika. The board consists of a 240×128 pixel monographic graphics display on one side, and as shown in Figure 4.11, a PIC18F8722 microcontroller on the other side. The device includes a RGB backlight and Touch Screen panel and circuit, and is operated from an 8 MHz crystal.

SmartGLCD has the following features:

- PIC18F8722 microcontroller;
- RA6963 display controller;
- Touch Screen controller;
- contrast potentiometer;
- FTDI chip (USB to RS232 converter);
- miniUSB connector;
- mikroSD card slot;
- reset button;
- programming interface.

www.it-ebooks.info

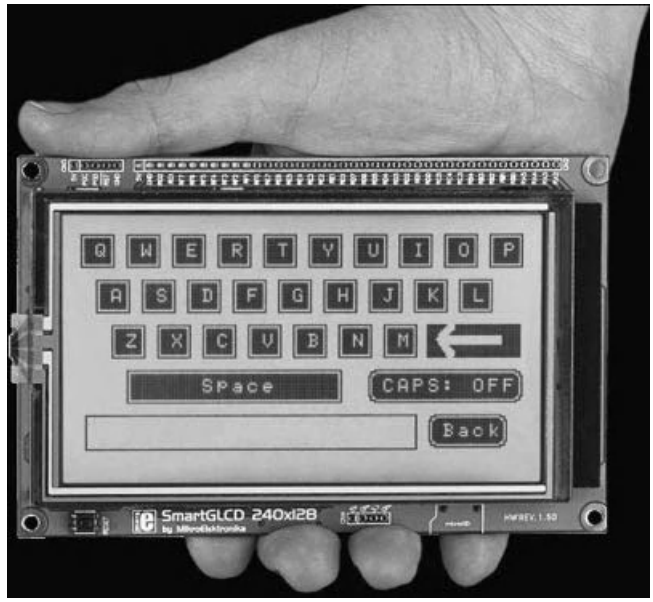


Figure 4.10 SmartGLCD front panel

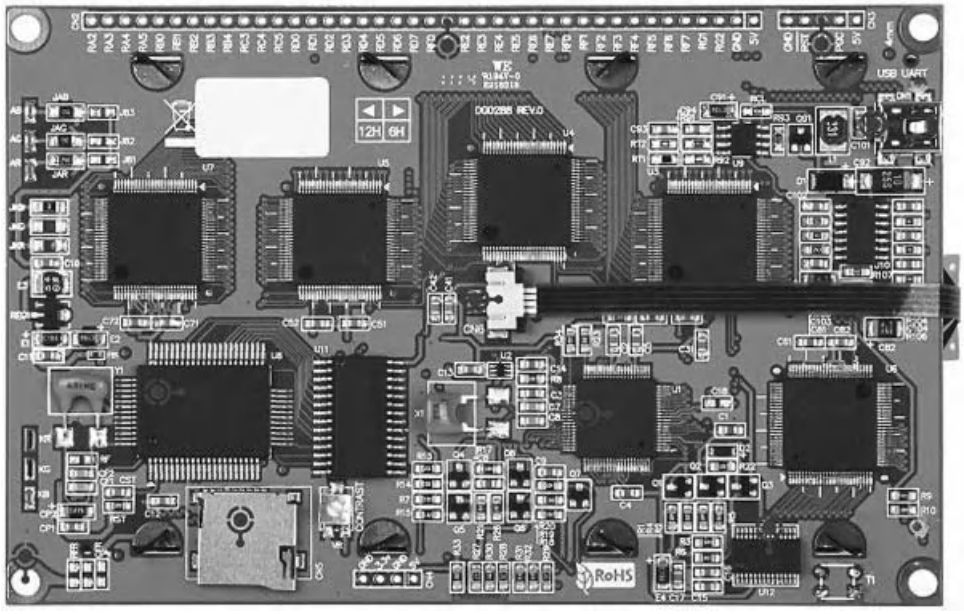


Figure 4.11 SmartGLCD back panel

www.it-ebooks.info

The SmartGLCD board can easily be programmed using the VisualGLCD software developed by mikroElektronika, together with a PIC microcontroller compiler of the company (e.g. mikroC Pro for PIC). We shall see how to program this board in later chapters.

4.2.1.2 Mikromedia Display Boards

Mikromedia display boards are manufactured by mikroElektronika. These boards are available for various types of microcontroller chips. The board for the PIC18 series of microcontrollers is known as the *Mikromedia for PIC18FJ* (or *MikroMMB*). This is a small (8×6 cm) full colour handheld TFT type display board with 320×240 pixels (see Figure 4.12). The other side of the board contains a PIC18F87J50 microcontroller and additional control and interface circuitry.

The board (see Figure 4.13) has the following features:

- PIC18F87J50 microcontroller;
- mini USB connector;
- reset button;
- audio module with headphone connector;
- Touch Screen controller;
- accelerometer chip;
- ICD2/ICD3 programming connectors;
- MicroSD card slot;
- Li-Polymer battery connector.

The Mikromedia for PIC18FJ board can easily be programmed using the VisualTFT software developed by mikroElektronika, together with a PIC microcontroller compiler of the

Figure 4.12 Mikromedia for PIC18FJ board

www.it-ebooks.info

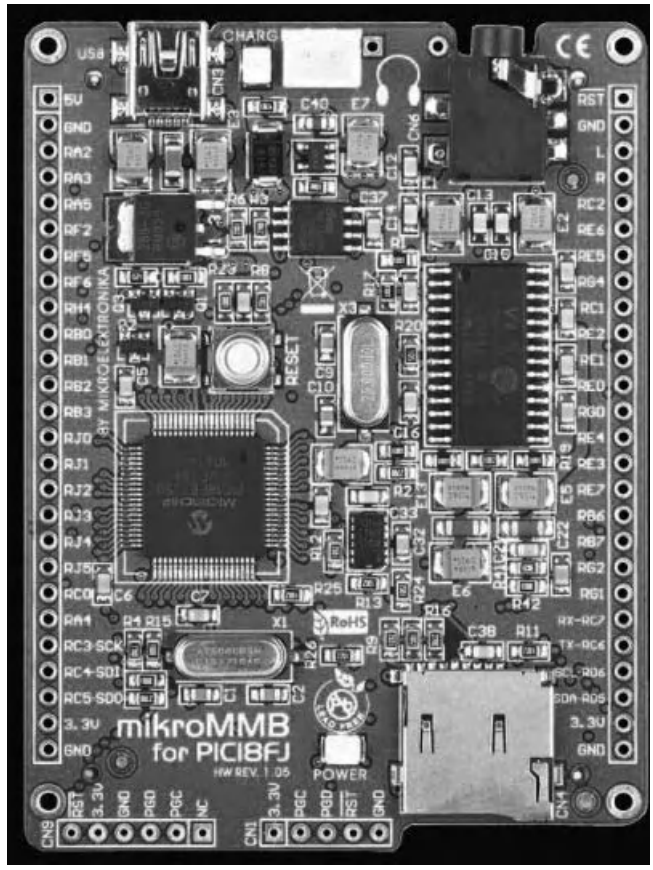


Figure 4.13 Component side of the board

company (e.g. mikroC Pro for PIC). We shall see how to program this board in the next chapters.

4.2.2 Display Software Tools

Display tools can be divided into three, depending on their complexity:

- font creation tools.
- display library tools.
- visual display tools.

4.2.2.1 Font Creation Tools

Font creation tools are used in both text based and GLCDs. In text based LCDs, users can create special symbols and characters. For example, mikroC Pro for PIC compiler includes a font creation tool to create fonts for 5×7 and 5×10 pixel text based LCDs. Figure 4.14

www.it-ebooks.info

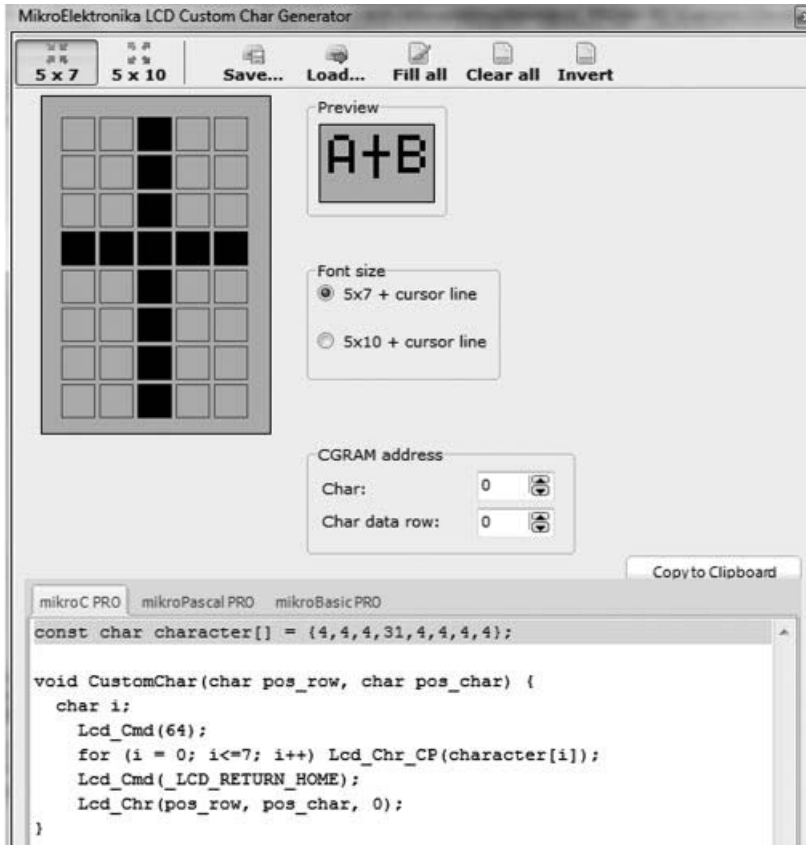


Figure 4.14 Creating fonts with the mikroC Pro for PIC compiler

shows an example symbol created using the mikroC Pro for PIC compiler. The compiler also generates the corresponding code that can be used in our program.

4.2.2.2 Display Library Tools

Most compilers offer display libraries where users can call these libraries, for example to display characters on an LCD, or to display shapes or graphs on GLCDs. We shall see examples in later chapters on how to use the mikroC Pro for PIC LCD and GLCD libraries. For example, we can display the text 'Hello' on the first row of an LCD with the mikroC Pro for PIC statement:

```
Lcd_Out(1, 1, 'Hello');
```

4.2.2.3 Visual Display Tools

The visual display tools are complex software packages that help the programmer to create GLCD based applications. Examples of such tools are the VisualGLCD and the VisualTFT software. Using these tools, for example, programmers can create real-time games, touch

www.it-ebooks.info

screen applications, and many more advanced real-time graphics applications. We shall see in later chapters how to use these tools in detail.

4.3 Using the In-Circuit Debugger with the EasyPIC7 Development Board

We shall be using the EasyPIC7 development board in some of our projects in later chapters. It is therefore worthwhile to look at an example and see how we can use the in-circuit debugger on this board.

In this example we will count up in binary and send the results to PORT C of the microcontroller, where we will see the LEDs connected to the port counting up in binary. The required program code is shown in Figure 4.15. The steps to use the in-circuit debugger are given below:

- Create a project, as described in Chapter 3 and write the program given in Figure 4.15. Select the microcontroller type as PIC18F45K22, and the clock frequency as 8 MHz.
- Compile the program, making sure that the *Build Type* is set to *ICD Debug* and the *Debugger* is set to *mikroICD* in the Project Settings window before the compilation (see Figure 4.16).

```
/*************************************************************************************************
                COUNTING UP IN BINARY
                -------------------------------------

This program counts up in binary and sends the results to PORT C of a PIC18F45K22
type microcontroller.

The easyPIC7 development board is used in this example, and the microcontroller is
operated from a 8MHz crystal

Author: Dogan Ibrahim
Date:   October, 2011
*************************************************************************************************/

void main()
{
unsigned char Cnt = 0;

TRISC = 0;                              // set direction to be output
for(;;)                                 // Do forever
{
        PORTC=Cnt;                      // send Cnt to PORT C
        Cnt++;                          // Increment Cnt
}
}
```

Figure 4.15 Program for the example

www.it-ebooks.info

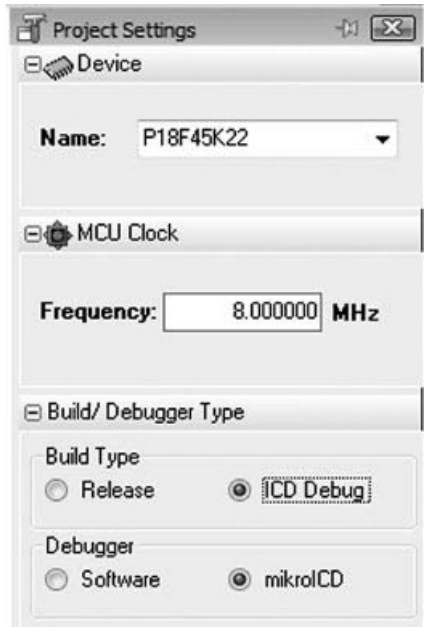


Figure 4.16 The Project Settings window

- Connect the EasyPIC7 board to the PC using the USB cable, making sure that the power supply jumper is set to 5 V. You should see the green power LED and the orange Link LED turning ON.
- Download your program to the program memory of the microcontroller. Click Tools -> mE Programmer (or press function key F11) from the drop-down menu. During the loading you will see the red Active LED turn ON, and the Blue Data LED will flash to indicate that the loading is in progress.
- Start the in-circuit debugger by clicking Run -> Start Debugger (or press function key F9). When the debugging is started, the program line, which will be next executed, is highlighted with a blue strip at the left-hand side of the window (see Figure 4.17). On the right-hand side you will see the debug window.
- Let us single step through the program and monitor the values of variable *Cnt*. To do this, select variable *Cnt* from the list box named *Select variable from list*. Then, click *Add* button in window *Watch Values*. The variable to be monitored can now be seen, as in Figure 4.18.
- Press function key F8 to single step through the program. You should see the value of *Cnt* incrementing. At the same time, the PORT C LED pattern on the EasyPIC7 development board will increment in binary every time data is sent to the port. Figure 4.19 shows the debugging process when *Cnt* is 4.
- Stop the debugger by clicking Run -> Stop Debugger (or function keys CTRL + F2).
- The in-circuit debugger supports breakpoints, and the way breakpoints are set and cleared are the same as when we were using the simulator (see Section 3.7).

www.it-ebooks.info

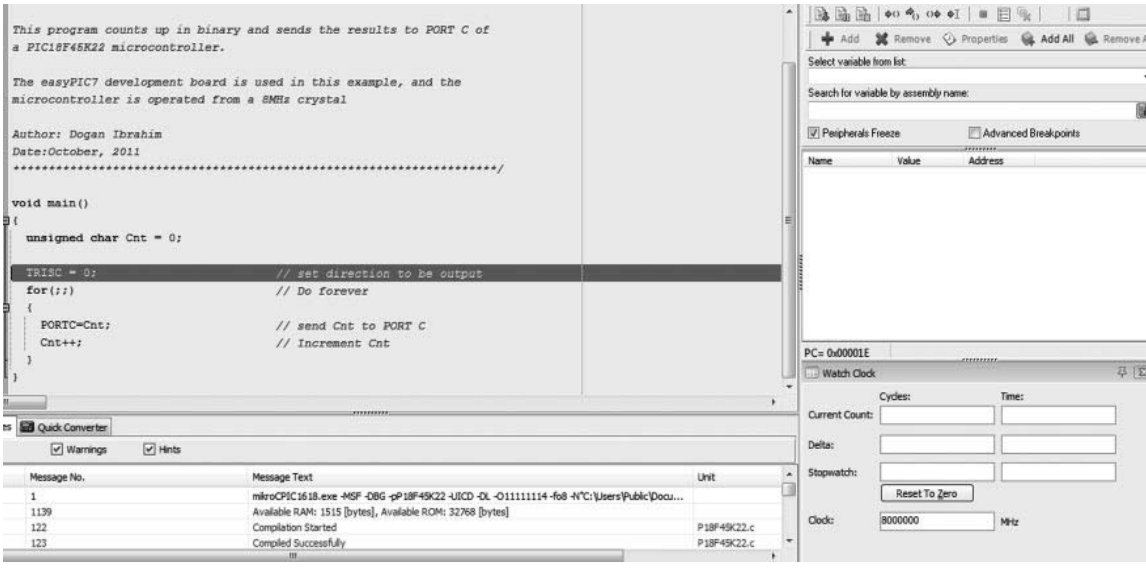


Figure 4.17 Program line to be executed next and the debug window

www.it-ebooks.info

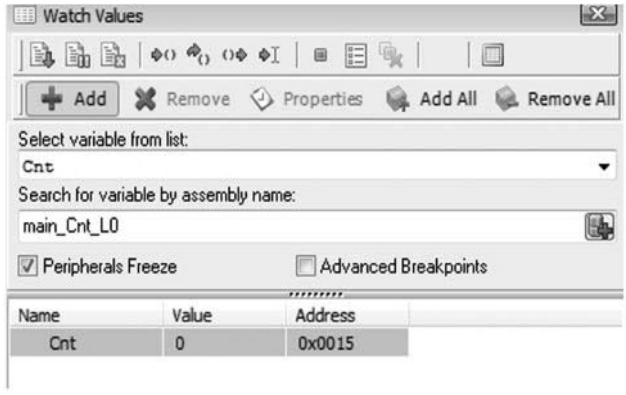


Figure 4.18 Monitoring variable Cnt

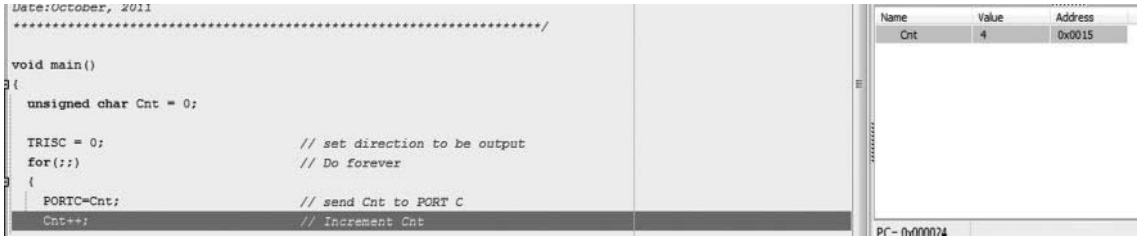


Figure 4.19 Debugging when Cnt is 4

www.it-ebooks.info

It is interesting to note that the new version of the mikroC Pro for PIC compiler (Version 5.2) supports advanced breakpoint options, such as placing breakpoints at specified addresses in the program memory, stopping the code execution when read/write access to the specified data memory location occurs, event breakpoints, and so on.

4.4 Summary

The microcontroller project development cycle has been explained in this chapter. In addition, the features of some of the commonly used microcontroller development boards have been described. Display development tools help the programmers to easily develop display based applications. These tools are described briefly in this chapter. Some of the projects in this book are based on using the EasyPIC7 microcontroller development board. An example use of this board is given where the on-board in-circuit debugger is used to step through a program loaded into the on-board microcontroller.

Exercises

4.1 Describe the microcontroller project development cycle with the help of a flowchart.
4.2 Explain the main differences between a simulator and a debugger.
4.3 Write a program to flash the LEDs connected to PORT C of a PIC microcontroller with 1 second intervals. Download the program to the EasyPIC7 development board and start the in-circuit debugger. Set the debugger to watch PORT C. Step through the program to see the LEDs flashing on the hardware. At the same time observe the values of PORT C in the debug window. Then, run the program continuously without the debugger and observe the LEDs flashing.

www.it-ebooks.info

5

Light Emitting Diodes (LEDs)

A LED is a tiny semiconductor light source, used mainly to indicate the status of electronic circuits, for example to indicate that power is applied to a circuit. The LED was invented in 1962 by Nick Holonyak while working at the General Electric Company. Early LEDs emitted low-intensity red light, but today high brightness and many colour LEDs are available. These LEDs are now used to replace incandescent and neon light bulbs in many energy efficient applications, for example LEDs are used as clusters in torches, automotive lights, traffic lights, signs and indicators, games, and so on. Infrared LEDs are used in most consumer remote control applications. LEDs offer many advantages over incandescent light sources, such as lower energy consumption, smaller size, longer lifetime, faster switching, available in many colours, low-voltage operation, do not get hot, and more robustness.

In this chapter we will review the various LED types used in microcontroller based applications, and also see how they can be used in electronic circuits. In addition, we will be looking at the basic principles of other commonly used LED based displays, such as single- and multi-digit 7-segment LEDs and alphanumeric LEDs.

5.1 A Typical LED

Figure 5.1 shows a typical LED with the electronic circuit symbol similar to that of a semiconductor diode. The device has two legs: the longer leg is the anode and the shorter leg the cathode. The cathode is also identified by a flat side on the body.

The intensity of the light emitted by an LED depends on the amount of forward current passed through the device. The maximum allowable forward current is denoted by I_{Fmax}. When designing an LED circuit, we have to know the typical voltage drop, V_{Typ} across the device, and the maximum allowable voltage drop, V_{Fmax}.

The brightness of the emitted light is measured in millicandela (mcd) and this is usually referenced to the forward current. For example, standard red LEDs are quoted to have brightness of 5 mcd when operated at 10 mA.

The viewing angle of a standard LED is about 60°, although some LEDs have viewing angles as narrow as 30°.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

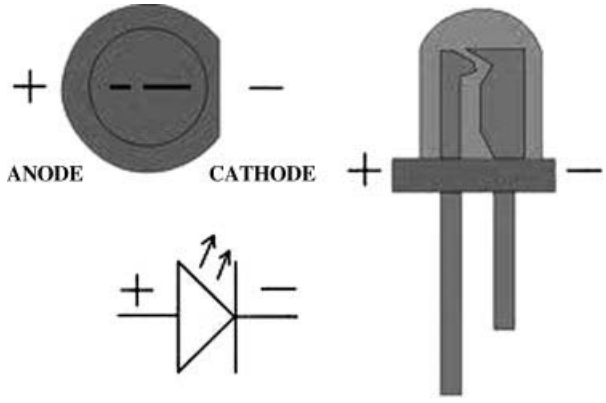


Figure 5.1 A typical LED

The colour of an LED depends on the type of chemical material used in its construction, and not on the colour of the glass enclosing the LED. As we shall see later, the colour of an LED is specified by giving the wavelength of the emitted light.

Most LEDs have typical forward voltage drop of 2 V, while blue and white ones can have as high as 4 V. The typical operating current is around 10 mA, although some small low-current LEDs can operate at around 1 mA. The higher the forward current the brighter the LED becomes, but care should be taken not to exceed the specified maximum allowable forward current.

Figure 5.2 shows the connection of an LED to a microcontroller output port. The port output voltage can be assumed to be +5 V when the port is at logic HIGH. Assuming that the LED is to be operated with 10 mA forward current, and that it has a forward voltage drop of 2 V, we can easily calculate the value of the current limiting resistor as

$$R = \frac{5\text{V} - 2\text{V}}{10\,\text{mA}} = 0.3\,\text{K} \tag{5.1}$$

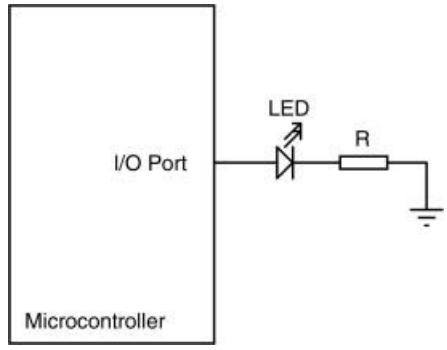


Figure 5.2 LED to microcontroller interface

www.it-ebooks.info

The nearest resistor we can choose is 330 Ω. It is interesting to note that the I/O pin of a PIC microcontroller can provide up to 25 mA current and thus LEDs can be driven with higher currents if desired, in order to obtain more brightness.

LEDs can easily be connected in series in circuits. When calculating the required current limiting resistor in such applications, the total voltage drop across all the LEDs should be taken as the overall forward voltage drop. LEDs can also be connected in parallel. When the same types of LEDs are connected in parallel, the value of the current limiting resistor is calculated by doubling the forward current requirements (e.g. 20 mA) and keeping the same forward voltage drop (e.g. 2 V). When different LEDs are connected in parallel, the situation is more complex, as only the LED with the lowest voltage drop will light and the high current through this LED may destroy it. It is best to use separate current limiting resistors in such applications.

5.2 LED Colours

Early LEDs were available only in low intensity red colour. Today, with the use of different inorganic semiconductor materials, visible LEDs are available in red, orange, yellow, blue, green, violet, purple and white. In addition, non-visible infrared and ultraviolet LEDs are also available for specific applications.

Table 5.1 gives a list of the available LEDs with their emitted wavelengths, typical forward voltage drops and the typical materials used in their construction.

The white LED is constructed either by using three primary colour LEDs, red, green and blue and mixing them to obtain white light (called RGB based white LEDs); or alternatively, by converting the monochromatic blue light into white by using a phosphor material (called phosphor based white LEDs).

The technical specifications of some LED types are given in Table 5.2. The LED selection for an applications depends on the following factors:

- size;
- shape;
- colour of emitted light (wavelength);
- colour of the case;
- brightness of the light;
- viewing angle.

Table 5.1 LED colours

Colour	Typical Voltage Drop	Wavelength	Typical Material
Red	2.0 V	610–760 nm	GaAsP
Orange	2.0 V	590–610 nm	AlGaInP
Yellow	2.1 V	570–590 nm	GaAsP
Green	3.0 V	500–570 nm	InGaN
Blue	3.0 V	450–500 nm	ZnSe
Violet	3.0 V	400–450 nm	InGaN
White	3.5 V	Broad	Blue + phosphor

www.it-ebooks.info

Table 5.2 Some LED specifications

Type	Colour	I_{Fmax}	V_{Fmax}	Intensity	Wavelength
Standard	Red	30 mA	2.1 V	5mcd@10 mA	660 nm
Standard	Bright red	30 mA	2.5 V	80cmd@10 mA	625 nm
Super bright	Red	30 mA	2.5 V	500cmd@20 mA	660 nm
Standard	Green	25 mA	2.5 V	35cmd@10 mA	565 nm
Bright	Blue	30 mA	5.5 V	60cmd @20 mA	430 nm

5.3 LED Sizes

The two most common LED sizes are known as T-1 and T-1 3/4. T-1 is small 3 mm diameter device, while T-1 3/4 is standard 5 mm diameter device. LEDs are also available in tiny SMD format, mounted on PCBs, and as right-angle devices for mounting on a PCB. Special brackets are available for mounting on panels or as boxes.

5.4 Bi-Colour LEDs

A bi-colour LED can be in two different forms: 2-pin and 3-pin. In a 2-pin LED, two different colour LEDs are combined in one package and wired in 'inverse parallel', that is one forwards, one backwards (see Figure 5.3). As shown in the truth table, only one LED is ON at any time, depending on the voltage applied to the pins.

In a 3-pin bi-colour LED, two different colour LEDs are usually connected in inverse, as shown in Figure 5.4. The common cathode pin is normally connected to the ground. Only one or both LEDs can be turned ON by supplying voltages to the anodes, as shown in the truth table.

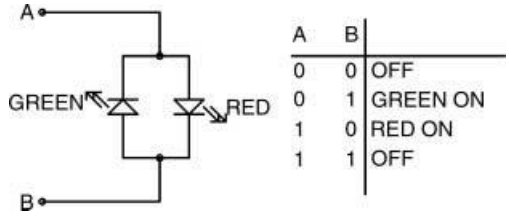


Figure 5.3 Bi-colour 2-lead LED

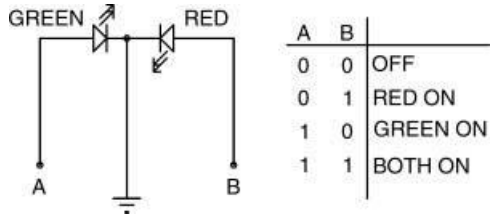


Figure 5.4 Bi-colour 3-lead LED

www.it-ebooks.info

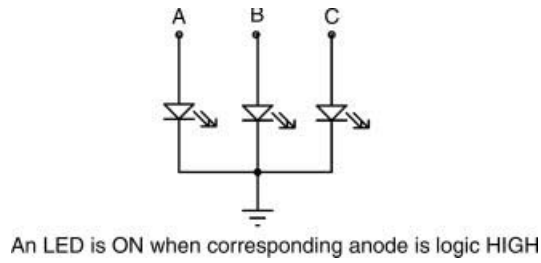


Figure 5.5 Tri-colour LED

5.5 Tri-Colour LEDs

These LEDs have 4 leads with 3 LEDs combined in one package, with common cathode or common anode configurations. Any number of LEDs can be turned ON by applying voltages to the pins. Figure 5.5 shows an example tri-colour LED with common cathode.

5.6 Flashing LEDs

A flashing LED is a single LED with 2 leads that flashes at a constant rate when a voltage is connected to its terminals. All the flashing circuitry and the current limiting resistor are included inside the LED assembly and the LED can be directly connected to a power supply. The flashing rate varies between different manufacturers, for example a typical flashing LED operates with 3.5 to 14 V and flashes approximately twice a second.

5.7 Other LED Shapes

LEDs are also available in different shapes and sizes. Figure 5.6 shows an LED assembly in the form of a bar, housed in a DIL package, and is available in different colours.

Figure 5.7 shows an LED dot-matrix assembly consisting of 5 rows and 7 columns of LEDs arranged as dots, and housed in a 14-pin DIL package. This type of dot-matrix LEDs is available in various colours, sizes and shapes. Several such LED assemblies can be connected to each other and a microcontroller is commonly used to display characters or symbols on the LED assembly.

Figure 5.6 LED bar

www.it-ebooks.info

Figure 5.7 LED dot-matrix assembly

Figure 5.8 Large LED dot-matrix display

LED dot-matrix displays are also used as large fixed or moving displays, usually in advertisements, or as signs. An example large display is shown in Figure 5.8.

5.8 7-Segment LEDs

7-segment displays were the earliest LED type electronic displays used to display numbers. These devices are commonly used in digital clocks and watches, electronic meters, electronic counting devices, and other equipment for displaying numeric only data.

A 7-segment LED consists of 7 light emitting elements arranged in a rectangular enclosure, and by turning the appropriate segments ON and OFF we can obtain the numbers 0 to 9. Figure 5.9 shows a typical 1-digit 7-segment display. Optionally, a decimal point is available to display fractional non-integer numbers. The segments of the displays are referred to by letters 'a' to 'g'.

7-segment displays are available in two configurations: common anode and common cathode. As shown in Figure 5.10, in a common anode connection, the anode pins of all the segments are connected together and this pin is usually connected to the power supply. Individual segments are turned ON by grounding the required segment pin. Similarly,

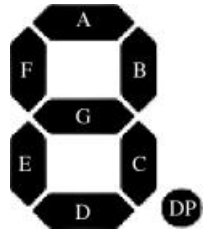


Figure 5.9 Typical 7-segment display

www.it-ebooks.info

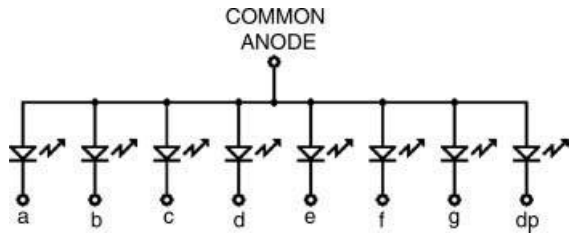


Figure 5.10 Common anode 7-segment LED

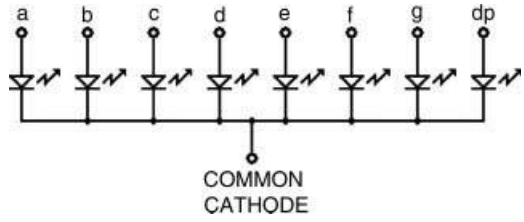


Figure 5.11 Common cathode 7-segment LED

Figure 5.11 shows a common cathode display. Here, the cathodes of all the segments are connected together and this pin is usually connected to ground. Individual segments are turned ON by applying voltage to the required segment pin.

5.8.1 Displaying Numbers

Figure 5.12 shows the connection between a microcontroller and a 7-segment display. As with standard LEDs, it is required to use current limiting resistors in each segment of the display to limit the current. Usually, resistor packages are used for ease of construction, lower cost and to save space. These packages consist of 8 or 10 same value resistors in a SIL

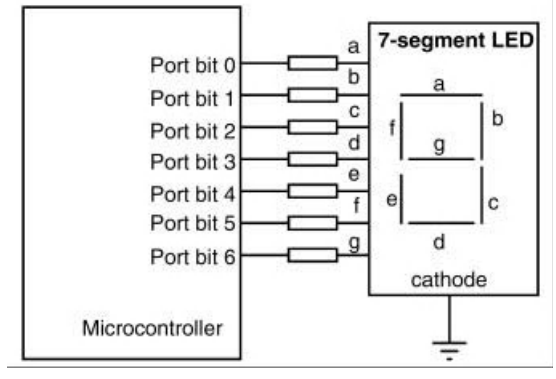


Figure 5.12 Connecting a 7-segment display to a microcontroller

www.it-ebooks.info

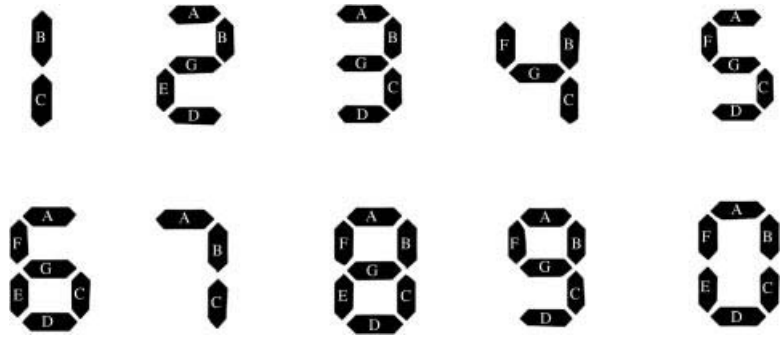


Figure 5.13 Displaying numbers 0 to 9

(Single-In-Line) type package with a common pin that can be grounded or connected to +V, depending on whether a common cathode or common anode display is used, respectively.

Figure 5.13 shows how the numbers 0 to 9 can be displayed by a 7-segment display. Notice that some symbols and lowercase letters (e.g. 'b', 'c', 'd', 'o', etc.) can also be displayed. In the early days of LEDs, 7-segment displays were used to display hexadecimal numbers from '0' to '9' and 'a' to 'f'.

The easiest way to display a number on the 7-segment LED is first to create a table showing the numbers and corresponding segments that should be turned ON or OFF to display the required number. Then, the bit pattern (or hexadecimal equivalent) required to display a number can be determined and the required number can be displayed by sending this bit pattern to the device. Table 5.3 shows the 7-segment LED encoding where numbers, the corresponding segment status, and the hexadecimal numbers required to be sent to the port where the display is connected to in order to display a specific number, are given. In constructing this table, it is assumed that the 8th bit of the microcontroller is 0, as it is not used by the display. For example, to display number 5, we have to send the hexadecimal number $0 \times 6D$ to the port that the display is connected to.

Table 5.3 7-segment LED encoding

Number	x g f e d c b a	Hexadecimal
0	0 0 1 1 1 1 1 1	3F
1	0 0 0 0 0 1 1 0	06
2	0 1 0 1 1 0 1 1	5B
3	0 1 0 0 1 1 1 1	4F
4	0 1 1 0 0 1 1 0	66
5	0 1 1 0 1 1 0 1	6D
6	0 1 1 1 1 1 0 1	7D
7	0 0 0 0 0 1 1 1	07
8	0 1 1 1 1 1 1 1	7F
9	0 1 1 0 1 1 1 1	6F

www.it-ebooks.info

Figure 5.14 A 2-digit 7-segment display

5.8.2 Multi-digit 7-Segment Displays

A 1-digit 7-segment display can only show numbers 0 to 9. In many applications, segments are joined together to create larger displays. A 2-digit display can show numbers between 0 and 99, a 3-digit between 0 and 999, a 4-digit between 0 and 9999, and so on.

Figure 5.14 shows a typical 2-digit 7-segment display. The digits are normally multiplexed to save I/O pins. For example, without multiplexing, a 2-digit display will require 14 pins, a 3-digit display will require 21 pins, and so on. With multiplexing, a 2-digit display will require only 9 pins, a 3-digit display will require 10 pins, and so on. Another advantage of multiplexing 7-segment LEDs is to reduce the power consumption considerably.

In multiplexed applications, all the digit segments are driven in parallel at the same time, but only the common pin (e.g. anode or cathode) of the required digit is enabled. The digits are enabled and disabled so fast that it gives the impression to the eye that both displays are ON at the same time. For example, suppose that we wish to display the number '25' on a 2-digit common cathode display. The steps are given below:

1. Send data to display '2' on both digits.
2. Enable the left (MSD) digit by grounding its cathode pin.
3. Wait for a while.
4. Send data to display '5' on both digits.
5. Enable the right (LSD) digit by grounding its cathode pin.
6. Wait for a while.
7. Go back to step 1.

The common pins of each digit are usually controlled using transistors switches. Figure 5.15 shows how a 2-digit display can be connected to a microcontroller using npn transistors to control the segment lines. Notice that setting the base of a transistor to logic HIGH will turn the transistor ON and hence will enable the common cathode pin connected to it.

5.9 Alphanumeric LEDs

Alphanumeric LED displays are used to display numerical as well as character data. These displays are similar to 7-segment LEDs, but are usually made up of 14-segments (plus a decimal point). Figure 5.16 shows a typical alphanumeric LED display. The segments are labelled 'a' to'm'. As with the 7-segment displays, numbers and letters are displayed by turning the appropriate segments ON and OFF. The internal structure of a typical 14-segment common anode alphanumeric display is shown in Figure 5.17.

www.it-ebooks.info

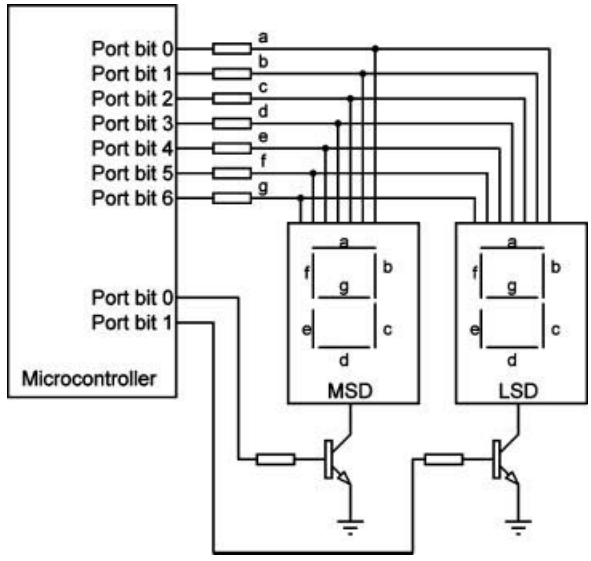


Figure 5.15 Connecting a 2-digit display to a microcontroller

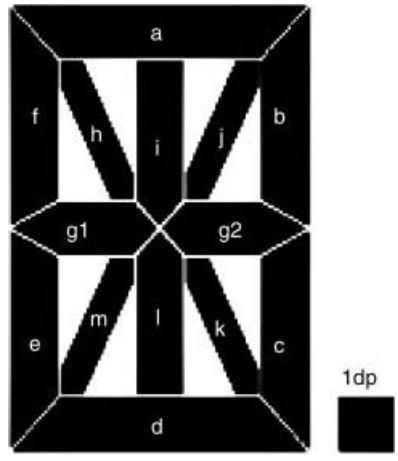


Figure 5.16 A typical 14-segment alphanumeric LED display

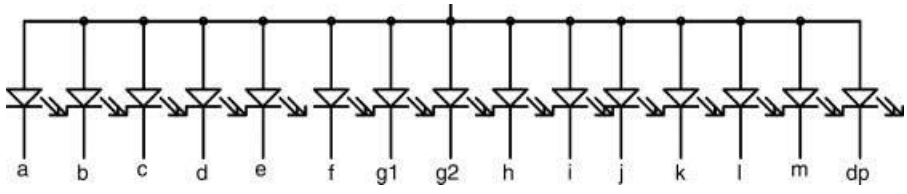


Figure 5.17 Inside a 14-segment common anode alphanumeric LED display

www.it-ebooks.info

Figure 5.18 shows how the numbers, letters and various symbols can be displayed on 14-digit alphanumeric LED displays. Alphanumeric LEDs are connected to microcontrollers in the same as the way 7-segment displays are connected.

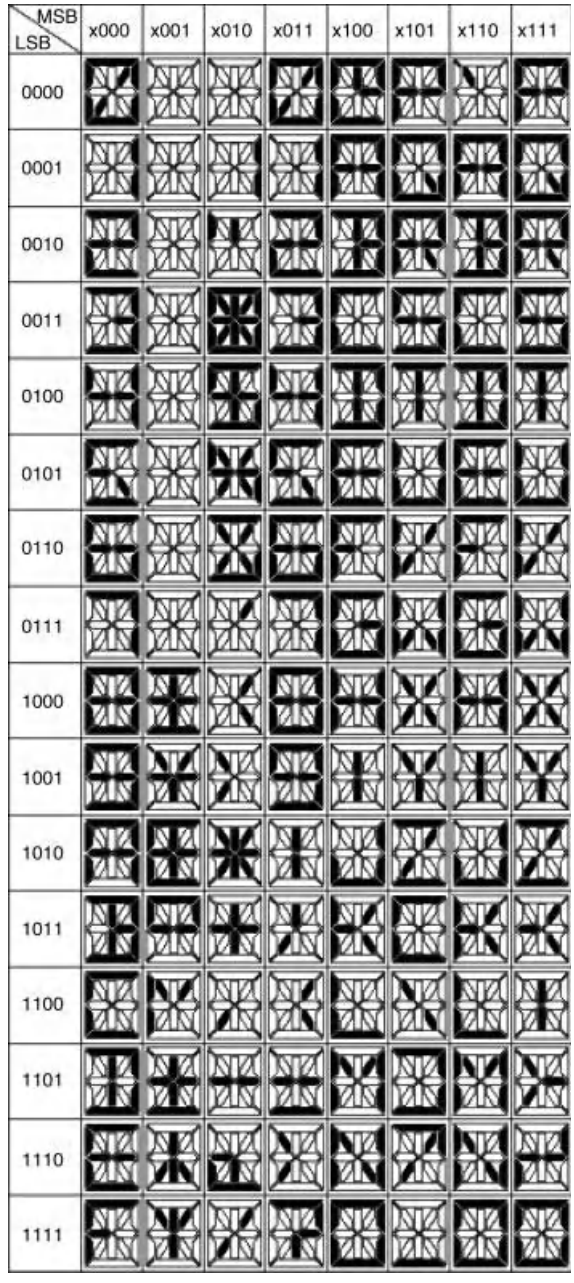


Figure 5.18 Displaying numbers, letters and symbols on alphanumeric displays

www.it-ebooks.info

A table similar to Table 5.3 can be constructed to determine how to display various characters. Table 5.4 shows what hexadecimal code to send to the port where the display is connected to, in order to display numbers '0' to '9' and letters 'A' to 'Z', assuming that the display is connected to a microcontroller using 14 I/O pins (this will normally be done using an 8-bit I/O port and another 6-bit I/O port), and bit 0 of the port is connected to segment 'a' of the display. For example, the letter 'A' is displayed by sending the hexadecimal code 0×04F7 to the display port.

Table 5.4 14-segment alphanumeric LED decoding

Number of Letter	m l k j I h g2 g1 f e d c b a	Hexadecimal
0	1 0 0 1 0 0 0 0 11 1 1 1 1	243F
1	00 0 0 0 0 0 0 0 00 0 0 1 1	0003
2	00 0 0 0 0 11 01 1 0 1 1	00DB
3	00 0 0 0 0 1 0 00 1 1 1 1	008F
4	00 0 0 0 0 11 1 0 0 1 1 0	00E6
5	00 1 0 0 0 0 1 10 1 0 0 1	0869
6	00 0 0 0 0 11 1 1 1 1 0 1	00FD
7	00 0 0 0 0 0 0 0 00 0 1 1 1	0007
8	00 0 0 0 0 11 1 1 1 1 1 1	00FF
9	00 0 0 0 0 11 10 1 1 1 1	00EF
A	0 0 0 1 0 0 11 11 0 1 1 1	04F7
B	01 0 01 0 1 0 00 1 1 1 1	128F
C	00 0 0 0 0 0 0 11 1 0 0 1	0039
D	0 1 0 0 1 0 0 0 0 0 1 1 1 1	120F
E	0 0 0 0 0 0 11 11 1 0 0 1	00F9
F	0 0 0 0 0 0 0 1 11 0 0 0 1	0071
G	0 0 0 0 0 0 1 0 11 1 1 0 1	00BD
H	0 0 0 0 0 0 11 11 0 1 1 0	00F6
I	0 1 0 01 0 0 0 0 0 0 0 0 0	1200
J	0 0 0 0 0 0 0 0 0 1 1 1 1 0	001E
K	0 0 1 1 0 0 0 1 11 0 0 0 0	0C70
L	0 0 0 0 0 0 0 0 11 1 0 0 0	0038
M	0 0 0 1 0 1 0 0 11 0 1 1 0	0536
N	0 0 1 0 0 1 0 0 11 0 1 1 0	0936
O	0 0 0 0 0 0 0 0 11 1 1 1 1	003F
P	0 0 0 0 0 0 11 11 0 0 1 1	00F3
Q	0 0 1 0 0 0 0 0 11 1 1 1 1	083F
R	0 0 1 0 0 0 11 11 0 0 1 1	08F3
S	0 0 0 0 0 0 11 10 1 1 0 1	0ED
T	0 1 0 0 1 0 0 0 0 0 0 0 0 1	1201
U	0 0 0 0 0 0 0 0 11 1 1 1 0	003E
V	10 0 1 0 0 0 0 11 0 0 0 0	2430
W	10 1 0 0 0 0 0 11 0 1 1 0	2436
X	10 1 1 0 1 0 0 0 0 0 0 0 0	2D00
Y	0 1 0 1 0 1 0 0 0 0 0 0 0 0	1500
Z	10 0 1 0 0 0 0 0 0 1 0 0 1	2409

www.it-ebooks.info

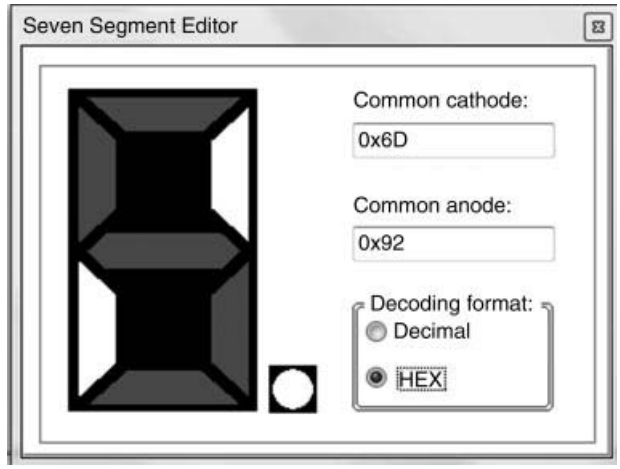


Figure 5.19 mikroC Pro for PIC 7-segment LED editor

Alphanumeric LED displays are usually multiplexed to display multi-digit numbers and texts. Multiplexing reduces the I/O pin requirements, overall power consumption and the number of connections considerably. The operating principles of multiplexed alphanumeric displays are the same as the standard multiplexed 7-segment displays.

Some companies offer multiplexed alphanumeric displays with integrated controllers, all housed on a small PCB and controlled with only a few serial lines. This simplifies the development of alphanumeric LED based projects considerably.

5.10 mikroC Pro for PIC 7-Segment LED Editor

The mikroC Pro for PIC language includes a 7-segment LED editor that can be used to find the decimal or hexadecimal codes to be sent to a port to display a given pattern. Patterns are created by clicking on the LED segments. The editor is invoked from the drop-down menu by clicking Tools -> Seven Segment Editor. Figure 5.19 shows a typical display of the editor where the pattern for number '5' is created. The code for this pattern is displayed as $0\times6D$ for common cathode displays (same code as in Table 5.3). Code for both common anode and common cathode displays can easily be obtained.

5.11 Summary

This chapter has described the basic principles of LED displays and showed how they can be used in microcontroller based projects.

The use of 7-segment LED displays has been described in detail, including the multiplexed multi-digit displays. The 7-segment encoding table is given that will enable the user to display any number by simply sending the correct code to the port where the display is connected to. The operation of dot-matrix LED displays have been explained briefly.

The chapter has also described the widely used LED alphanumeric displays, used to display numbers 0 to 9, letters A to Z, and some common symbols. The encoding table of

www.it-ebooks.info

alphanumeric displays is given to enable the user to display any required number or letter by reading from the table and sending the correct code to the display port.

Exercises

5.1 Explain the differences between different colour LEDs.
5.2 What are the important parameters of LED displays?
5.3 Explain what the LED 'viewing angle' is, and why is it important?
5.4 Draw a circuit diagram to show how an LED can be used in a microcontroller based project. Why is there need for a current limiting resistor? How can you calculate the value of this resistor?
5.5 What are the advantages of bi-colour and tri-colour LEDs? Give one example use for each LED type.
5.6 Draw a circuit diagram to show how a 7-segment LED display can be connected to a microcontroller.
5.7 Explain how number '8' can be displayed on a 7-segment display.
5.8 Use the mikroC Pro for PIC 7-segment editor to create the pattern for number '0'. What is the hexadecimal code for this pattern in both common anode and common cathode configurations?
5.9 Explain how 7-segment LED displays can be multiplexed. Draw the circuit diagram to show a 4-digit multiplexed display. Explain how the number '2364' can be displayed with such a display.
5.10 Explain the differences between 7-segment and alphanumeric LED displays.
5.11 Draw the circuit diagram to show how an alphanumeric display can be connected to a microcontroller. Explain how you would display letter 'Z' on such a display.
5.12 Explain how alphanumeric LED displays can be multiplexed. Draw the circuit diagram to show how a 2-digit alphanumeric display can be constructed. Explain how the letters 'XY' can be displayed with such a display.

www.it-ebooks.info

6

Liquid Crystal Displays (LCDs) and mikroC Pro for PIC LCD Functions

Displays are an important part of most microcontroller based applications. A video display for example, would make a microcontroller application much more user-friendly, as it allows text messages and images to be displayed in real-time. Standard video displays require complex interfaces and are large and costly. The alternatives to video displays are the LCDs (and graphics LCDs). These devices come in different shapes and sizes. Some LCDs have only one row, while others can have up to four rows. Some have back lighting so that the display can be viewed in dimly lit conditions.

There are basically two types of LCDs as far as the interfacing method is concerned: parallel LCDs and serial LCDs. Parallel LCDs (e.g. Hitachi HD44780) are the most commonly used ones and they are connected to microcontrollers using four to eight data lines and some control lines. Serial LCDs are connected to microcontrollers using only one data line and data is sent to the LCD using the RS232 serial communications protocol. Serial LCDs are easier to use, but they cost a lot more than the parallel ones.

In this book we are interested in the parallel LCDs only, as they are the ones used commonly in display applications. In this chapter we shall be looking at the basic operating principles of these LCDs and see how they can be used and programmed in microcontroller based applications. In addition, we shall be looking at the mikroC Pro for PIC LCD library functions, which simplify the development of LCD based applications considerably.

6.1 HD44780 Controller

HD44780 is perhaps the most popular LCD controller module used in microcontroller projects and it is currently the industry standard LCD module. This module is monochrome and comes in different shapes and sizes. Depending upon the requirements, displays with 8, 16, 20, 24, 32 and 40 characters are available. The row size can be selected as 1, 2 or 4. Display

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

Figure 6.1 1 × 16 LCD display

types are identified by specifying the number of rows and number of characters per row. For example, a 1 × 16 display (see Figure 6.1) has one row with 16 characters, and a 4 × 16 display has 4 rows and 16 characters on each row (see Figure 6.2).

The LCD normally has 14 pins for connection to the outside world. The pins are usually organised in a single row and numbered 1 to 14, as shown in Figure 6.3. Those with backlights have two additional pins. Table 6.1 shows the pin configuration. The device is normally operated from a voltage +3.3 to +5 V.

Vss is the ground pin. The Vdd pin should be connected to a positive supply. Although the manufacturers specify a +5 V supply, the device will work with as low as +3 V or as high as +6 V.

Pin 3 is named the Vee and this is the contrast adjustment pin. This pin is used to adjust the contrast of the display and it should be connected to a variable voltage supply. A 10 K potentiometer is normally connected between the power supply lines with its wiper arm connected to this pin so that the contrast can be adjusted as desired. Figure 6.4 shows a typical connection of this pin.

Pin 4 is the Register Select (RS) and when this pin is LOW, any data sent to the display is treated as commands. When RS is HIGH, data sent is treated as character data for the display.

Pin 5 is the Read/write (R/W) line. This pin is pulled LOW in order to write commands or character data to the display (i.e. microcontroller to display data transfer). When this pin is HIGH, character data or status information can be read from the display module (i.e. display to microcontroller data transfer). The R/W pin is usually connected to ground, as we normally want to send commands and data to the display.

Pin 6 is the Enable (E) or clock pin used to initiate the transfer of command or data to the display. When writing to the display, data is transferred on the HIGH to LOW transition of

Figure 6.2 4 × 16 LCD display

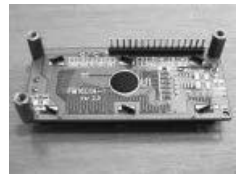


Figure 6.3 LCD pins

www.it-ebooks.info

Table 6.1 LCD pin configuration

Pin no	Name	Function
1	Vss	Ground
2	Vdd	+V supply
3	Vee	Contrast adjustment
4	RS	Register select
5	R/W	Read/write
6	E	Enable (clock)
7	D0	Data bit 0
8	D1	Data bit 1
9	D2	Data bit 2
10	D3	Data bit 3
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7
15 (optional)	B+	Backlight +
16 (optional)	B−	Backlight −

this pin. Similarly, when reading from the display, data becomes available after the LOW to HIGH transition of this pin.

Pins 7 to 14 are the eight data bus lines (D0 to D7). As we shall see later, data transfer between the microcontroller and the LCD can take place using either 8-bit byte, or two 4-bit nibbles. In the latter case, only the upper 4 bits of the data bus pins (D4 to D7) are used and a byte is transferred in two successive operations. The advantage of using 4-bit mode is that fewer I/O lines are required to communicate with the LCD.

The LCD can display all of the standard ASCII characters. In addition, some symbols can also be displayed. Characters are made up of either 5×7 or 5×10 dots. Figure 6.5 shows

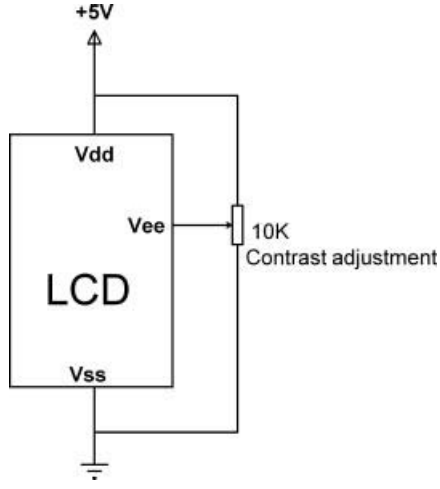


Figure 6.4 Adjusting the display contrast

www.it-ebooks.info

Figure 6.5 LCD standard character set

the standard LCD character set, although different manufacturers can specify different character sets.

6.2 Displaying User Defined Data

Figure 6.6 shows the block diagram of the HD44780 LCD module. The module contains three types of memories: an 80×8 bits display data RAM (DDRAM), 64 bytes of character generator RAM (CGRAM), and a 9920 bit character generator ROM (CGROM). The data displayed currently by the LCD is stored in the DDRAM. On displays with fewer than 80 characters, any unused DDRAM locations can be used as a general purpose RAM. The CGRAM stores the user defined character set. The CGROM stores the fixed character set of the LCD, which can vary between different manufacturers. The device has two registers, an instruction register (IR) and a data register (DR). The IR can be written from the microcontroller and it stores the instruction codes and address information. The DR stores the data to be written into the DDRAM or the CGRAM. Data written into the DR is automatically transferred to the DDRAM or the CGRAM. An address counter is used to address the DDRAM and CGRAM. This counter is incremented or decremented automatically after writing (or reading) into the DDRAM or the CGRAM.

The user can specify any pattern of up to 8 characters made up of a 5×7 matrix, or up to 4 characters with a 5×10 matrix. Figure 6.7 shows the relationship between the character codes, CGRAM address and the CGRAM data for 5×7 display matrix. CGRAM address bits 3 to 5 (000 to 111) correspond to 8 characters. CGRAM address bits 0 to 2 (000 to 111) correspond to the line numbers of the characters. CGRAM data bits 0 to 4 correspond to the column data of the pattern to be generated. The user defined CGRAM characters are displayed when character code bits 4 to 7 are all 0. In the example in Figure 6.7, the user defined character 'R' will be displayed when character code 0×00 is selected (notice that since bit 3 of the character code is don't care, the character can also be displayed by selecting code 0×08).

www.it-ebooks.info

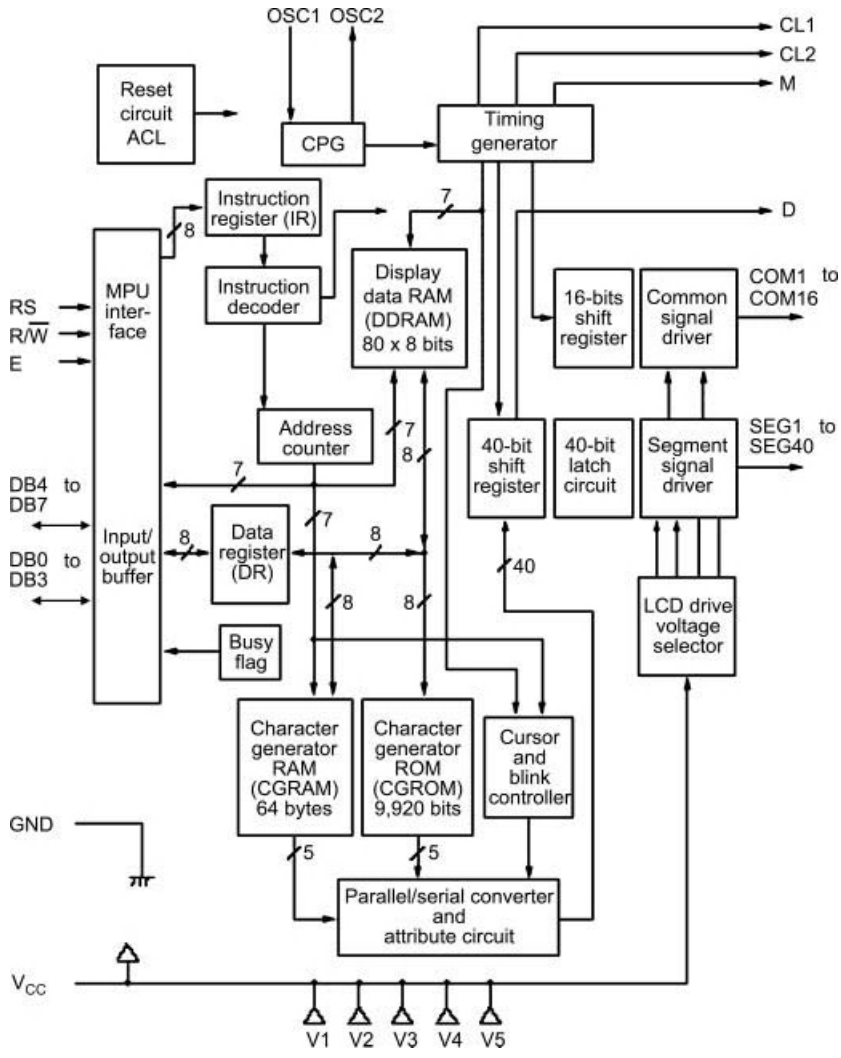


Figure 6.6 Block diagram of the HD44780 module

The CGRAM data for character 'R' in Figure 6.7 is given below. This data can be loaded into the CGRAM using the LCD instruction Set CGRAM Address and Write Data to CGRAM (see Sections 6.4.7 and 6.4.10):

```
0x1E, 0x11, 0x11, 0x1E, 0x14, 0x12, 0x11,0x0
```

6.3 DDRAM Addresses

The DDRAM addresses of various display sizes are given in this section. Table 6.2 gives the 1-row display DDRAM addresses.

www.it-ebooks.info

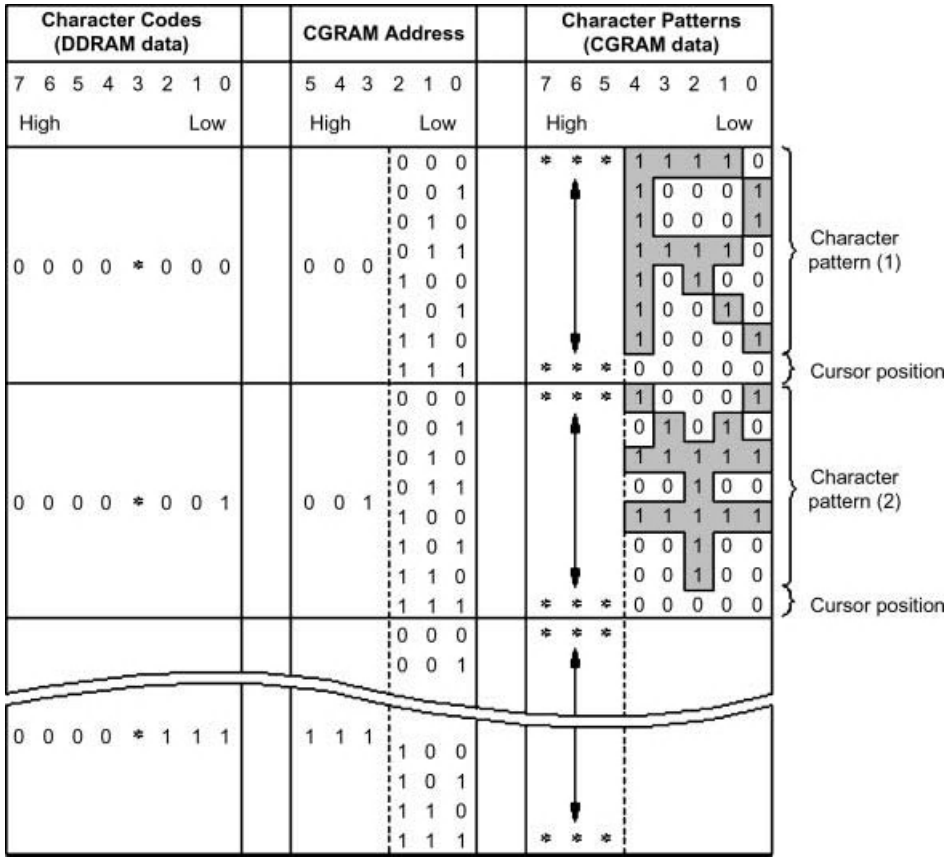

Character Codes (DDRAM data)
CGRAM Address
Character Patterns (CGRAM data)
7 6 5 4 3 2 1 0
High Low
5 4 3 2 1 0
7 6 5 4 3 2 1 0
0 0 0 0 * 0 0 0
0 0 0
0 0 0 0 * 0 0 1
0 0 1
0 0 0 0 * 1 1 1
1 1 1
Character pattern (1)
Cursor position
Character pattern (2)
Cursor position

Figure 6.7 Relationship between character codes, CGRAM addresses and CGRAM data

The 2-row display DDRAM addresses are given in Table 6.3. Notice that the second row starts from address 0×40.

The 4-row display DDRAM addresses are given in Table 6.4. As can be seen from this table, the second row starts from address 0×40, the third row starts from address 0×14, and the fourth row starts from address 0×54. Two internal controllers are used for 4-row displays.

Table 6.2 1-row display DDRAM addresses

Display Size	Character Position	DDRAM Address
1×8	00 to 07	0×00 to 0×07
1×16	00 to 15	0×00 to $0 \times 0F$
1×20	00 to 19	0×00 to 0×13
1×24	00 to 23	0×00 to 0×17
1×32	00 to 31	0×00 to $0 \times 1F$
1×40	00 to 39	0×00 to 0×27

www.it-ebooks.info

Table 6.3 2-row display DDRAM addresses

Display Size	Character Position	DDRAM Address
2 × 16	00 to 15	0 × 00 to 0 × 0F and 0 × 40 to 0 × 4F
2 × 20	00 to 19	0 × 00 to 0 × 13 and 0 × 40 to 0 × 53
2 × 24	00 to 23	0 × 00 to 0 × 17 and 0 × 40 to 0 × 57
2 × 32	00 to 31	0 × 00 to 0 × 1F and 0 × 40 to 0 × 5F
2 × 40	00 to 39	0 × 00 to 0 × 27 and 0 × 40 to 0 × 67

Table 6.4 4-row display DDRAM addresses

Display Size	Character Position	DDRAM Address
4 × 16	00 to 15	0 × 00 to 0 × 0F and 0 × 40 to 0 × 4F and 0 × 14 to 0 × 23 and 0 × 54 to 0 × 63
4 × 20	00 to 19	0 × 00 to 0 × 13 and 0 × 40 to 0 × 53 and 0 × 14 to 0 × 27 and 0 × 54 to 0 × 67
4 × 40	00 to 39	0 × 00 to 0 × 27 and 0 × 40 to 0 × 67 on both controllers

6.4 Display Timing and Control

Although the mikroC Pro for PIC compiler includes a library of functions for using LCD displays, it is worthwhile to learn the basic principles of how the LCD operates.

Correct operation of the LCD display requires knowledge of the instruction set and correct timing. Figure 6.8 shows LCD the instruction set. The instructions are described in the next sections.

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Time
Clear Display	0	0	0	0	0	0	0	0	0	1	1.64ms
Return Home	0	0	0	0	0	0	0	0	1	*	1.64ms
Entry Mode set	0	0	0	0	0	0	0	1	I/D	S	40µs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	40µs
Cursor/Display shift Function set	0	0	0	0	0	1	S/C	R/L	*	*	40µs
Function Set	0	0	0	0	1	DL	N	F	*	*	40µs
Set CGRAM	0	0	0	1	A	A	A	A	A	A	40µs
Set DDRAM	0	0	1	A	A	A	A	A	A	A	40µs
Read Busy Flag	0	1	BF	A	A	A	A	A	A	A	0
Write to CGRAM or DDRAM	1	0	D	D	D	D	D	D	D	D	40µs
Read from CGRAM or DDRAM	1	1	D	D	D	D	D	D	D	D	40µs

Figure 6.8 HD44780 instruction set

www.it-ebooks.info

6.4.1 Clear Display

This instruction clears the display and returns the cursor to the home position. ASCII 'Space' codes (hexadecimal 0×20) are filled in all DDRAM addresses and the address counter is reset to 0. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

6.4.2 Return Cursor to Home

This instruction returns the cursor to the home position, which is the top left corner of the display. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

6.4.3 Cursor Move Direction

This instruction sets the cursor move direction and specifies whether or not to shift the display. These operations are performed during data read and write. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

The DDRAM address is incremented (I/D = 1) or decremented (I/D = 0) by 1 when a character code is written into or read from the DDRAM. In normal operations, I/D is set to 1 to increment the cursor and move it to the right after writing a character.

Bit S shifts the entire display to the right (when I/D = 0) or left (when I/D = 1) when set to 1.

6.4.4 Display ON/OFF

This instruction sets the display ON or OFF. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

The display is ON when D = 1, and OFF when D = 0.
The cursor displays when C = 1, and does not display when C = 0.
When B = 1, the character indicated by the cursor blinks.

www.it-ebooks.info

6.4.5 Cursor and Display Shift

This instruction moves the cursor and shifts the display without changing the DDRAM contents. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	*	*

In a 2-row display, the cursor moves to the second line when it passes the end of the first row. S/C and R/L control the shift operations as follows:

S/C	R/L	
0	0	Shift cursor to the left
0	1	Shift cursor to the right
1	0	Shift the entire display to the left
1	1	Shift the entire display to the right

6.4.6 Function Set

This instruction sets the interface data length (DL), number of display rows (N), and the character font (F). The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	D/L	N	F	*	*

When DL = 1, the DL is set to 8 bits; when DL = 0, the DL is set to 4-bits.
N sets the number of display lines (1 or 2)
F sets the character font; F = 0 for 5×7 font, and F = 1 for 5×10 font.

6.4.7 Set CGRAM Address

This instruction sets the CGRAM address. CGRAM data is sent and received after this setting. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	A	A	A	A	A	A

After this instruction the address counter is loaded with address AAAAAA. Data is then written to or read from the microcontroller for the CGRAM.

6.4.8 Set DDRAM Address

This instruction sets the DDRAM address. DDRAM data is sent and received after this setting. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

www.it-ebooks.info

After this instruction, the address counter is loaded with address AAAAAAA. Data is then written to or read from the microcontroller for the DDRAM.

Notice that for 1-line displays, AAAAAAA is from 00 to $0 \times 4F$. For 2-line displays, AAAAAAA is from 00 to 0×27 for the first line, and from 0×40 to 0×67 for the second line.

6.4.9 *Read Busy Flag*

This instruction reads the busy flag (BF), which indicates whether or not the LCD module is ready to receive commands or character data. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	A	A	A	A	A	A	A

BF = 1 indicates that internal operation is in progress and the LCD module cannot receive new commands or character data. The instruction also returns value of the address counter in AAAAAAA.

6.4.10 *Write Data to CGRAM or DDRAM*

This instruction writes data into DDRAM or CGRAM. The 8-bit data DDDDDDDD is written to the CGRAM or the DDRAM. The memory to be written is determined by the previous specifications of CGRAM or DDRAM address setting. The instruction format is:

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D	D	D	D	D	D	D	D

6.4.11 *Read Data from CGRAM or DDRAM*

This instruction is used to read data from the CGRAM or the DDRAM. The previous specification of CGRAM or DDRAM determines whether the CGRAM or the DDRAM is to be read.

6.5 LCD Initialisation

The LCD controller has to be initialised before it can be used. The initialisation sequence is different when the LCD is operated in 4-bit or 8-bit mode. Although the 4-bit mode is more commonly used, we will look at the initialisation of the LCD when operated in both modes.

www.it-ebooks.info

6.5.1 8-bit Mode Initialisation

The steps to initialise the LCD in 8-bit mode are as follows:

1. Wait at least 15 ms after power is applied.
2. Write 0×030 to LCD and wait 5 ms for the instruction to complete.
3. Write 0×030 to LCD and wait 160 μs for the instruction to complete.
4. Write 0×030 to LCD and wait 160 μs for the instruction to complete.
5. Set the operating characteristics of the LCD:
   - set interface length (8-bit);
   - turn off display;
   - turn on display;
   - set entry mode.

6.5.2 4-bit Mode Initialisation

The steps to initialise the LCD in 4-bit mode are

1. Wait at least 15 ms after power is applied.
2. Write 0×03 to LCD and wait 5 ms for the instruction to complete.
3. Write 0×03 to LCD and wait 160 μs for the instruction to complete.
4. Write 0×030 to LCD and wait 160 μs for the instruction to complete.
5. Set the operating characteristics of the LCD:
   - Write 0×02 to set 4-bit mode.
     Following instructions/data writes must be in two nibbles
   - Set interface length and font.
   - Turn off display.
   - Turn on display and enable blink cursor.
   - Set entry mode.

In most LCD based projects, the 4-bit mode is used. Considering only the 4-bit mode of operation with 5×7 character matrix, 2-lines, and cursor blink ON, the initialisation sequence can be described in detail as follows (notice that the data is sent in the upper byte only, i.e. DB4 to DB7):

1. Wait at least 15 ms after power is applied.
2. Set RS = 0, R/W = 0.
3. Send 0×03 to LCD.

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	1

4. Set E = 1, followed by E = 0.
5. Wait 5 ms.
6. Send 0×03 to LCD.

www.it-ebooks.info

7. Set E = 1, followed by E = 0.
8. Wait 160 μs.
9. Send 0 × 03 to LCD.
10. Set E = 1, followed by E = 0.
11. Wait for 160 μs.
12. Send 0 × 02 to LCD to set 4-bit mode.

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	1	0

13. Send E = 1, followed by E = 0.
14. Wait 40 μs.
15. Send 0 × 02 to LCD (for 4-bit mode, 5 × 8 font).
16. Set E = 1, followed by E = 0.
17. Send 0 × 08 to LCD.

RS	R/W	DB7	DB6	DB5	DB4
0	0	1	0	0	0

18. Set E = 1, followed by E = 0.
19. Wait 40 μs.
20. Send 0 × 00 to LCD (display OFF).
21. Set E = 1, followed by E = 0.
22. Send 0 × 08 to LCD.
23. Set E = 1, followed by E = 0.
24. Wait 40 μs.
25. Send 0 × 00 to LCD (display ON, blink ON).
26. Set E = 1, followed by E = 0.
27. Send 0 × 0F to LCD.
28. Set E = 1, followed by E = 0.
29. Wait 40 μs.
30. Send 0 × 00 to LCD (set entry mode).
31. Set E = 1, followed by E = 0.
32. Send 0 × 06 to LCD.

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	1	1	0

33. Set E = 1, followed by E = 0.
34. Wait 40 μs.
35. Set RS = 1.
36. End of initialisation.

In steps 15 to 18, the byte 0 × 28 is sent in two nibbles to the display. That is, D/L = 0, N = 1 and F = 0, which sets the interface to 4-bit mode, 2-lines, and 5 × 7 font.

www.it-ebooks.info

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	D/L	N	F	*	*

In steps 20 to 23, the byte 0×08 is sent in two nibbles to turn OFF the display. That is, $D = 0$, $C = 0$ and $B = 0$, which sets display OFF, cursor OFF, and blink OFF.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

In steps 25 to 28, the byte $0 \times 0F$ is sent in two nibbles to turn the display ON. That is, $D = 1$, $C = 1$ and $B = 1$, which turns the display ON, turns the cursor ON, and turns ON the cursor blink.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	D	C	F	*	*

In steps 30 to 33, the byte 0×06 is sent in two nibbles to set the entry mode. That is, I/D = 1 and S = 0, which is set to increment the cursor after each byte is written to the display, and display shift is disabled.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

The minimum pulse width of the 'E' clock line is specified as 450 ns by the manufacturers.

6.6 Example LCD Display Setup Program

An example C program is given here to show how the LCD can be set up. The connection of the LCD to the microcontroller is shown in Figure 6.9 (if you are using the EasyPIC 7 development board, all you have to do is just connect the LCD to the board). PORT B lower pins

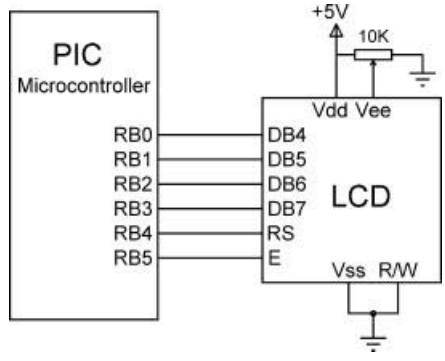


Figure 6.9 Connecting the LCD to a PIC microcontroller

www.it-ebooks.info

(RB0 to RB3) are connected to the upper 4 bits of the LCD (DB4 to DB7). In addition, RB4 and RB5 pins are connected to LCD RS and E pins, respectively.

The program listing is shown in Figure 6.10. The following functions are used in the program:

```
/*********************************************************************************************
                        LCD DISPLAY ROUTINES
                        ------------------------------------

This program implements LCD routines. The LCD is connected to a PIC microcontroller
 as follows:

 LCD Pin      Microcontroller
  DB4            RB0
  DB5            RB1
  DB6            RB2
  DB7            RB3
  RS             RB4
  E              RB5

The easyPIC7 development board is used in this example, and the microcontroller is
operated from a 8MHz crystal

Author:       Dogan Ibrahim
Date:         October, 2011
*********************************************************************************************/

#define LCD_RS PORTB.F4
#define LCD_E PORTB.F5

//
// This function toggles the E input of the LED
//
void Toggle_E(void)
{
   LCD_E = 1;
   LCD_E = 0;
}


//
// This fucntion sends a command to the LCD
//
void Lcd_Write_Cmd(unsigned char c)
{
   PORTB = (PORTB & 0xF0) | (c >> 4);                    // Send upper nibble
   Toggle_E();
   PORTB = (PORTB & 0xF0) | (c & 0x0F);                  // Send lower nibble
   Toggle_E();
   Delay_us(40);
}


//
// This function initializes the LCD
//
```

Figure 6.10 LCD setup and display program

www.it-ebooks.info

```
void Lcd_Initialize(void)
{
   Delay_Ms(15);
   LCD_RS = 0;
   PORTB = 0x03;
   Toggle_E();
   Delay_Ms(5);
   Toggle_E();
   Delay_us(160);
   Toggle_E();
   Delay_us(160);
   PORTB = 0x02;                          // In 4-bit mode
   Toggle_E();
   Delay_us(40);
   Lcd_Write_Cmd(0x28);                   // 4-bit mode, 5x7 font
   Lcd_Write_Cmd(0x08);                   // Display OFF
   Lcd_Write_Cmd(0x0F);                   // Display ON, cursor blink
   Lcd_Write_Cmd(0x06);                   // Set entry mode
}


//
// This function writes a character to the LCD
//
void Lcd_Write_Char(char c)
{
   LCD_RS = 1;                            // In data mode
   PORTB = (PORTB & 0xF0) | (c >> 4);     //  Send upper nibble
   Toggle_E();
   PORTB = (PORTB & 0xF0) | (c & 0x0F);   // Send lower nibble
   Toggle_E();
   Delay_us(40);
}

//
// This function clears the LCD
//
 Lcd_Clear(void)
{
   LCD_RS = 0;                            // In command mode
   Lcd_Write_Cmd(0x01);                   // Write a command
   Delay_Ms(2);
}


//
// This is the main program to test the LCD
//
void main()
{
   char Txt[] = "Hello";                  // Text to be displayed
```

Figure 6.10 (*Continued*)

www.it-ebooks.info

```
    char i;

    TRISB = 0;                                  // PORT B is output
    Lcd_Initialize();                           // Initialize LCD
    Lcd_Clear();                                // Clear LCD

    for(i = 0; i < 5; i++)Lcd_Write_Char(Txt[i]);   // Send characters to LCD
}
```

Figure 6.10 (*Continued*)

Lcd_Initialise: This function initialises the LCD in 4-bit mode, as described earlier. The enable clock pulses are sent to the LCD by calling to function Toggle_E, which simply sets the E line to HIGH and then LOW. The 4-bit LCD commands are sent using function Lcd_Write_Cmd.

Lcd_Write_Cmd: This function receives a byte, separates it into two nibbles, sends the high nibble, followed by the low nibble. The E line is toggled after each output.

Lcd_Clear: This function clears the LCD and homes the cursor.

Lcd_Write_Char: This function sends a character to the LCD. The upper nibble is sent first, followed by the lower nibble. Line E is toggled between each output.

Toggle_E: This function toggles the E line by first sending a HIGH and then a LOW signal.

The LCD setup functions are tested by the main program in Figure 6.10. Here, PORT B pins are configured to be output. Then the LCD is initialised by calling Lcd_Initialise and LCD is cleared by calling function Lcd_Clear. The text 'Hello' is then sent to the LCD by using function Lcd_Write_Char, where each character is sent individually using a *for* loop.

6.7 mikroC Pro for PIC LCD Functions

The mikroC Pro for PIC language provides a library of LCD functions for the HD44780 type controller. The library functions are based on using 4-bit interface. A list of the offered functions and their usage are described in this section.

6.7.1 Lcd_Init

This function initialises the LCD module and it must be called before any of the other LCD functions are called. The function is called with no arguments. Before the function is called, the interface between the microcontroller and the LCD must be defined using statements of the following format:

```
//Lcdpinout settings
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D4 at RB0_bit;
```

www.it-ebooks.info

```
//Pin direction
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D7_Direction at TRISB3_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D4_Direction at TRISB0_bit;
```

The configuration above assumes that the connection between the LCD and the microcontroller is as follows:

LCD	Microcontroller Port
RS	RB4
EN	RB5
D7	RB3
D6	RB2
D5	RB1
D4	RB0

Example call: `Lcd_Init();`

6.7.2 Lcd_Out

This function displays text on the LCD starting from specified row and column positions. Both string variables and literals can be passed as a text.

Example call: `Lcd_Out(1, 3, "Hello"); //Display text "Hello" at row 1, column 3`

6.7.3 Lcd_Out_Cp

This function displays text at the current cursor position. Both string variables and literals can be passed as text.

Example call: `Lcd_Out_Cp("Hello"); //Display text "Hello" at current position`

6.7.4 Lcd_Chr

This function displays a single character at the specified row and column positions. Both variables and literals can be passed as a character.

Example call: `LcdChrt(1, 2, 'X'); //Display character "X" at row 1, column 2`

6.7.5 Lcd_Chr_Cp

This function displays a single character at the current cursor position. Both variables and literals can be passed as a character.

www.it-ebooks.info

Table 6.5 Valid LCD commands

LCD Command	Purpose
_LCD_FIRST_ROW	Move cursor to the 1st row
_LCD_SECOND_ROW	Move cursor to the 2nd row
_LCD_THIRD_ROW	Move cursor to the 3rd row
_LCD_FOURTH_ROW	Move cursor to the 4th row
_LCD_CLEAR	Clear display
_LCD_RETURN_HOME	Return cursor to home position, returns a shifted display to its original position. DDRAM is unaffected.
_LCD_CURSOR_OFF	Turn off cursor
_LCD_UNDERLINE_ON	Underline cursor on
_LCD_BLINK_CURSOR_ON	Blink cursor on
_LCD_MOVE_CURSOR_LEFT	Move cursor left without changing DD RAM
_LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing DD RAM
_LCD_TURN_ON	Turn LCD display on
_LCD_TURN_OFF	Turn LCD display off
_LCD_SHIFT_LEFT	Shift display left without changing DDRAM
_LCD_SHIFT_RIGHT	Shift display right without changing DD RAM

Example call: `Lcd_Chr_Cp('X'); //Display character "X" at current position`

6.7.6 Lcd_Cmd

This function sends a command to the LCD. A list of the valid commands is given in Table 6.5.

Example call: `Lcd_Cmd(_LCD_CLEAR); //Clear display`

The mikroC Pro for PIC LCD functions will be used in Chapter 11 when we develop LCD based projects.

6.8 Summary

This chapter has briefly described the operating principles of LCD displays when used in microcontroller based circuits.

The creation of user defined characters has been described with an example.

The LCD instructions and the LCD initialisation sequence have been given for both 4-bit and 8-bit interfaces. A C program is given to show how the LCD can be initialised from the first principles and how it can be used in a program.

mikroC Pro for PIC language provides a library of LCD functions that can be extremely useful while developing LCD based microcontroller projects. A list of the available functions is given, with examples to show how they can be called from C programs.

www.it-ebooks.info

Exercises

6.1 Draw a circuit diagram to show the typical connection of an LCD to a microcontroller when the LCD is used in 4-bit interface mode.
6.2 Explain the functions of the LCD pins.
6.3 Describe how a user defined character can be created for an LCD. What data should be loaded into the CGRAM to create a ‘ + ’ symbol?
6.4 Describe the steps required to initialise an LCD for operation in 4-bit mode.
6.5 What are the advantages of using an LCD in 4-bit mode?
6.6 What are the advantages of using an LCD in 8-bit mode?
6.7 Draw a flowchart to show the steps required to initialise an LCD in 4-bit mode.
6.8 Modify the program given in Figure 6.10 by adding a function to home the cursor.
6.9 What LCD functions are available in the mikroC Pro for PIC language? Show the program code required to display text ‘Computer’, starting from column 3 of the second row.
6.10 Write the mikroC Pro for PIC program code required to clear only the top row of a 2×16 LCD.
6.11 Write a mikroC Pro for PIC program code to display texts ‘LED’ and ‘LCD’ in rows 1 and 2 of a 2×16 LCD, respectively.

www.it-ebooks.info

7

Graphics LCD Displays (GLCD)

Graphics LCD displays (GLCDs) are commonly used in many scientific applications, where we may want to display graphical data, such as a bar-chart or *x*-*y* line graph, for example a graph showing the change of temperature over time, and so on. GLCDs are also used in many consumer applications, such as mobile phones, MP3 players, GPS systems, games, educational toys, and so on. Another important application area of GLCDs is in industrial automation and control, where various plant characteristics can easily be monitored or changed.

Most GLCD applications nowadays make use of touch-screen facilities. As we shall see later, a touch screen is basically a transparent layer with touch sensitive cells, placed on top of a standard GLCD. When the user touches the screen, the screen co-ordinates of the touched point are sent to the processor. For example, nearly all mobile phone manufacturers offer touch-screen phones, where users can easily navigate between different screens and make choices by simply touching on the displayed items, thus avoiding the use of buttons for screen navigation.

There are several GLCD screens and GLCD controllers in use currently. For small applications, the 128×64 pixel monochrome GLCD with the KS0108 controller is one of the most commonly used displays. For larger display requirements and more complex projects, we can select the 240×128 pixel monochrome GLCD screen with the T6963 (or RA6963) controller. For colour GLCD based applications, TFT type displays seem to be the best choice currently.

In this chapter we shall be looking at how the standard 128×64 GLCD can be interfaced and used in microcontroller based projects. In addition, the principles and use of the touch-screen displays will be described in this chapter. The use of 240×128 pixel GLCDs and TFT based colour GLCDs will be described in later chapters, when we develop display based projects.

7.1 The 128×64 Pixel GLCD

These GLCDs have dimensions 7.8×7.0 cm and a thickness of 1.0 cm. The viewing area is 6.2×4.4 cm. The display consists of 128×64 pixels, organised as 128 pixels in the horizontal direction and 64 pixels in the vertical direction. The display operates with +5 V supply, consumes typically 8 mA current, and comes with a built-in KS0108 type display controller.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

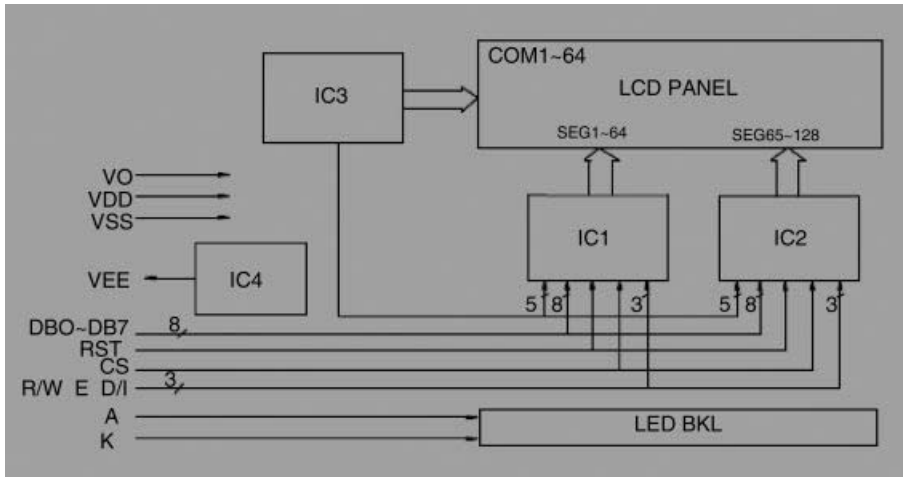


Figure 7.1 Block diagram of the GLCD with KS0108 controller

A backlight LED is provided for visibility in low ambient light conditions. This LED consumes about 360 mA when operated.

The block diagram of the GLCD is shown in Figure 7.1. Basically, two controllers are used internally: one for segments 1 to 64, and the other for segments 65 to 128.

The display is connected to the external world through a 20-pin SIL (Single-In-Line) type connector. Table 7.1 gives the pin numbers and corresponding pin names.

Table 7.1 128 × 64 pixel GLCD pin configuration

Pin No	Pin Name	Function
1	\CSAor CS1	Chip select for controller 1
2	\CSB or CS2	Chip select for controller 2
3	VSS	Ground
4	VDD	+5 V
5	V0	Contrast adjustment
6	D/I	Register select
7	R/W	Read-write
8	E	Enable
9	DB0	Data bus bit 0
10	DB1	Data bus bit 1
11	DB2	Data bus bit 2
12	DB3	Data bus bit 3
13	DB4	Data bus bit 4
14	DB5	Data bus bit 5
15	DB6	Data bus bit 6
16	DB7	Data bus bit 7
17	RST	Reset
18	VEE	Negative voltage
19	A	LED +4.2 V
20	K	LED ground

www.it-ebooks.info

The description of each pin is as follows:

/CSA, /CSB: Chip select pins for the two controllers. The display is logically divided into two sections and these signals control, which half should be enabled at any time.

VCC, GND: Power supply and ground pins

V0: Contrast adjustment. A 10 KB potentiometer should be used to adjust the contrast. The wiper arm should be connected to this pin, the other two arms should be connected to VEE and ground.

D/I: Register select pin. Logic HIGH is data mode, logic LOW is instruction mode

R/W: Read-write pin. Logic HIGH is read, logic LOW is write

E: Enable pin. Logic HIGH to LOW to enable

DB0 – DB7: Data bus pins

RST: Reset pin. The display is reset if this pin is held LOW for at least 100 ns. During reset the display is off and no commands can be executed by the display controller.

VEE: Negative voltage outputpin for contrast adjustment.

A, K: Power supply and ground pins for the backlight. Pin K should be connected to ground and pin A should be connected +5 V supply through a 10 ohm resistor.

Figure 7.2 shows connection of the GLCD to a microcontroller, with the contrast adjustment potentiometer and backlight LED connections also shown.

7.2 Operation of the GLCD Display

The internal operation of the GLCD display and the KS0108 controller are very complex and beyond the scope of this book. Most microcontroller compiler developers provide libraries for using these displays in their programming languages. mikroC Pro for PIC compiler provides a very advanced GLCD library, and we shall be looking at the details of the functions in this library in the next section. In this section, only the basic information required before using the GLCD library are given.

Figure 7.3 shows the structure of the GLCD, as far as programming the display is concerned. The 128×64 pixel display is logically split into two halves. There are two controllers: controller A controlling the left half of the display and controller B controlling the right half, where the two controllers are addressed independently. Each half of the display consists of 8 pages, where each page is 8 bits high and 8 bytes (64 bits) wide. Thus, each half consists of 64×64 bits. Text is written to the pages of the display. Thus, a total of 16 characters

www.it-ebooks.info

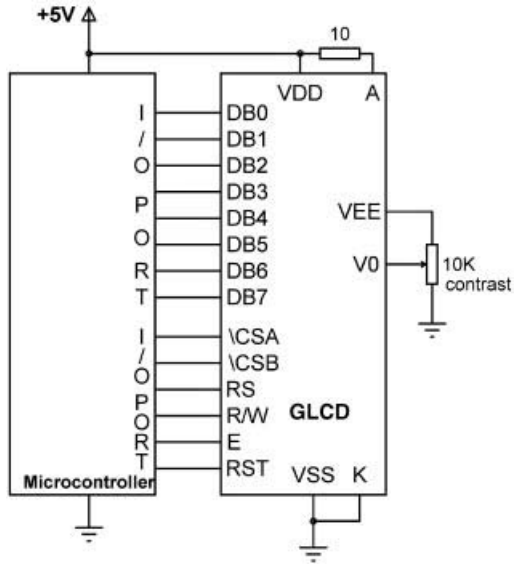


Figure 7.2 Connecting the 128 × 64 GLCD to a microcontroller

across can be written for a given page on both halves of the display. Considering that there are 8 pages, a total of 128 characters can be written on the display.

The origin of the display is the top left-hand corner (see Figure 7.4). The X-direction extends towards the right, and the Y-direction extends towards the bottom of the display. In

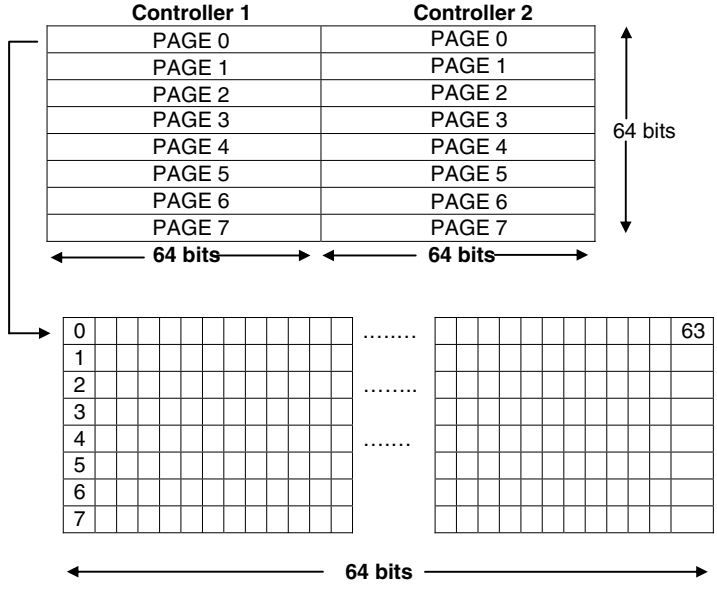


Figure 7.3 Structure of the GLCD

www.it-ebooks.info

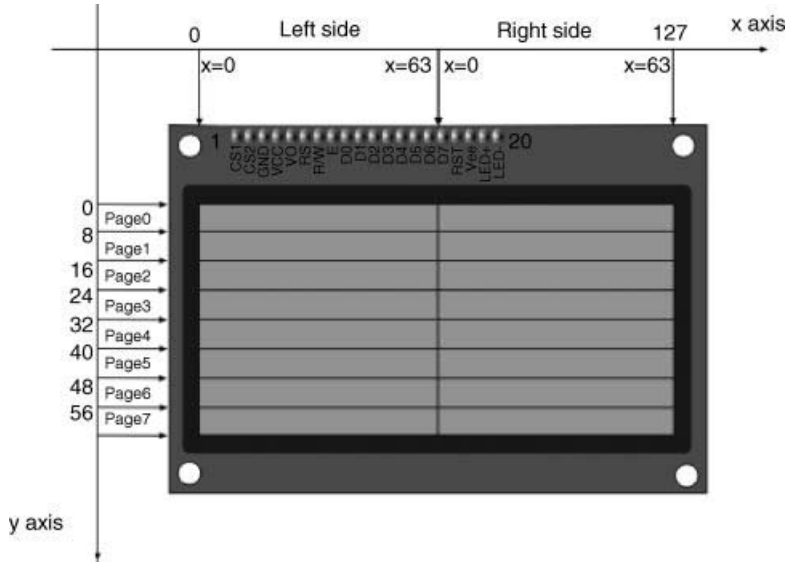


Figure 7.4 GLCD display coordinates

the X-direction, the pixels range from 0 to 127, while in the Y-direction, the pixels range from 0 to 63. Co-ordinate (127, 63) is at the bottom right-hand corner of the display.

7.3 mikroC Pro for PIC GLCD Library Functions

mikroC Pro for PIC language supports the 128 × 64 pixel GLCDs and provides a large library of functions for the development of GLCD based projects. In fact, there are libraries for several different types of GLCDs. In this section we shall be looking at the commonly used library functions provided for the 128 × 64 GLCDs, working with the KS0108 controller.

7.3.1 Glcd_Init

This function initialises the GLCD module. The function is called with no arguments and it must be called before any other GLCD functions are called.

The GLCD control and data lines can be configured by the user, but the 8 data lines must be on a single port. Before this function is called, the interface between the GLCD and the microcontroller must be defined using *sbit* type statements of the following format. In the following example, it is assumed that the GLCD data lines are connected to PORT D, and in addition the CS1, CS2, RS, RW, EN and RST lines are connected to PORT B:

```
// GLCDpinout settings
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS at RB2_bit;
```

www.it-ebooks.info

```
sbit GLCD_RW at RB3_bit;
sbit GLCD_EN at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbitGLCD_RS_Directionat TRISB2_bit;
sbitGLCD_RW_Directionat TRISB3_bit;
sbitGLCD_EN_Directionat TRISB4_bit;
sbitGLCD_RST_Directionat TRISB5_bit;

Example Call:       Glcd_Init();
```

7.3.2 *Glcd_Set_Side*

This function selects the GLCD side based on the argument, which is the x-co-ordinate. Values from 0 to 63 specify the left side of the display, while values from 64 to 127 specify the right side.

```
Example Call:    Glcd_Set_Side(0);    // Select left hand side of display
```

7.3.3 *Glcd_Set_X*

This function sets the *x*-axis position from the left border of the GLCD within the selected display side.

```
Example Call:       Glcd_Set_X(10);          // Set position to pixel 10
```

7.3.4 *Glcd_Set_Page*

This function selects a page of the GLCD. The argument to the function is the page number between 0 and 7.

```
Example Call:       Glcd_Set_Page(2);          // Select Page 2
```

7.3.5 *Glcd_Write_Data*

This function writes 1 byte to the current location on GLCD memory and moves to the next location. The GLCD side, and page number should be set before calling this function.

```
Example Call:       Glcd_Write_Data(MyData);
```

7.3.6 *Glcd_Fill*

This function fills the GLCD memory with the specified byte pattern, where the pattern is passed as an argument to the function.

```
Example Call:       Glcd_Fill(0);              // Clears the screen
```

www.it-ebooks.info

7.3.7 Glcd_Dot

This function draws a dot on GLCD at co-ordinates x_pos, y_pos. The x and y co-ordinates and the colour of the dot are passed as arguments. Valid x co-ordinates are 0 to 127, valid y co-ordinates are 0 to 63, and valid colours are 0 to 2, where 0 clears the dot, 1 places a dot, and 2 inverts the dot.

```
Example Call:    Glcd_Dot(0, 10, 1);        // Place a dot at x = 0, y = 10
```

7.3.8 Glcd_Line

This function draws a line on the GLCD. The arguments passed to the function are:

```
x_start:  x co-ordinate of the line starting position (0 to 127)
y_start:  y co-ordinate of the line starting position (0 to 63)
x_end:    x co-ordinate of the line ending position (0 to 127)
y_end:    y co-ordinate of the line ending position (0 to 63)
colour:   The colour value between 0 and 2. 0 is white, 1 is black, and 2
          inverts each dot.
```

```
Example Call:    Glcd_Line(0, 0, 5, 10, 1);        // Draw a line from (0,0)
                                                      to (5,10)
```

7.3.9 Glcd_V_Line

This function draws a vertical line on the GLCD. The arguments passed to the function are:

```
y_start:  y co-ordinate of the line starting position (0 to 63)
y_end:    y co-ordinate of the line ending position (0 to 63)
x_pos:    x co-ordinate of the vertical line (0 to 127)
colour:   The colour value between 0 and 2. 0 is white, 1 is black, and 2
          inverts each dot.
```

```
Example Call:    Glcd_V_Line(4, 10, 5, 1);        // Draw a line from (5,4)
                                                     to (5,10)
```

7.3.10 Glcd_H_Line

This function draws a horizontal line on the GLCD. The arguments passed to the function are:

```
x_start:  x co-ordinate of the line starting position (0 to 127)
x_end:    x co-ordinate of the line ending position (0 to 127)
y_pos:    y co-ordinate of the vertical line (0 to 63)
colour:   The colour value between 0 and 2. 0 is white, 1 is black, and 2
          inverts each dot.
```

www.it-ebooks.info

```
Example Call:    Glcd_H_Line(15, 55, 25, 1);      // Draw a line from
                                                     (15,25) to (55,25)
```

7.3.11 Glcd_Rectangle

This function draws a rectangle on the GLCD. The arguments passed to the function are:

```
x_upper_left:     x co-ordinate of the upper left corner of rectangle
                  (0 to 127)
y_upper_left:     y co-ordinate of the upper left corner of rectangle
                  (0 to 63)
x_bottom_right:   x co-ordinate of the lower right corner of rectangle
                  (0 to 127)
y_bottom_right:   y co-ordinate of the lower right corner of rectangle
                  (0 to 63)
colour:           The colour value between 0 and 2. 0 is white, 1 is black,
                  and 2 inverts each dot.

Example Call:     Glcd_Rectangle(5, 5, 10, 10);
                  //Draw rectangle between (5,5) and (10,10)
```

7.3.12 Glcd_Rectangle_Round_Edges

This function draws a rounded edge rectangle on the GLCD. The arguments passed to the function are:

```
x_upper_left:     x co-ordinate of the upper left corner of rectangle
                  (0 to 127)
y_upper_left:     y co-ordinate of the upper left corner of rectangle
                  (0 to 63)
x_bottom_right:   x co-ordinate of the lower right corner of rectangle
                  (0 to 127)
y_bottom_right:   y co-ordinate of the lower right corner of rectangle
                  (0 to 63)
round radius:     radius of the rounded edge
colour:           The colour value between 0 and 2. 0 is white, 1 is black,
                  and 2 inverts each dot.

Example Call:     Glcd_Rectangle_Round_Edge(5, 5, 10, 10, 15, 1);
                  // Draw rectangle between (5,5) and (10,10) with
                  edge radius 15
```

7.3.13 Glcd_Rectangle_Round_Edges_Fill

This function draws a filled rounded edge rectangle on the GLCD with colour. The arguments passed to the function are:

www.it-ebooks.info

```
x_upper_left:    x co-ordinate of the upper left corner of rectangle
                 (0 to 127)
y_upper_left:    y co-ordinate of the upper left corner of rectangle
                 (0 to 63)
x_bottom_right:  x co-ordinate of the lower right corner of rectangle
                 (0 to 127)
y_bottom_right:  y co-ordinate of the lower right corner ofrectangle
                 (0 to 63)
round radius:    radius of the rounded edge
colour:          colour of the rectangle border. The colour value is
                 between 0 and 2. 0 is white, 1 is black, and 2 inverts
                 each dot.

Example Call:    Glcd_Rectangle_Round_Edges_Fill(5, 5, 10, 10, 15, 1);
                 // Draw rectangle between (5,5) and (10,10) with
                  edge radius 15
```

7.3.14 Glcd_Box

This function draws a box on the GLCD. The arguments passed to the function are:

```
x_upper_left:    x co-ordinate of the upper left corner of box (0 to 127)
y_upper_left:    y co-ordinate of the upper left corner of box (0 to 63)
x_bottom_right:  x co-ordinate of the lower right corner of box (0 to 127)
y_bottom_right:  y co-ordinate of the lower right corner of box (0 to 63)
colour:          colour of the box fill.The colour value is between 0
                 and 2. 0 is white, 1 is black, 2 inverts each dot.

Example Call:    Glcd_Box(5, 15, 20, 30, 1);     // Draw box between
                                                    (5,15) and (20,30)
```

7.3.15 Glcd_Circle

This function draws a circle on the GLCD. The arguments passed to the function are:

```
x_center:  x co-ordinate of the circle center (0 to 127)
y_center:  y co-ordinate of the circle center (0 to 63)
radius:    radius of the circle
colour:    colour of the circle line. The colour value is between 0 and 2. 0
           is white, 1 is black, 2 inverts each dot.

Example Call:    Glcd_Circle(30, 30, 5, 1);
                 // Draw circle with center at (30,30), and radius 5
```

www.it-ebooks.info

7.3.16 Glcd_Circle_Fill

This function draws a filled circle on the GLCD. The arguments passed to the function are:

```
x_center:   x co-ordinate of the circle center (0 to 127)
y_center:   y co-ordinate of the circle center (0 to 63)
radius:     radius of the circle
colour:     The colour value is between 0 and 2. 0 is white, 1 is black, 2
            inverts each dot.
```

```
Example Call:    Glcd_Circle_Fill(30, 30, 5, 1);
                 // Draw a filled circle with center at (30,30), and radius 5
```

7.3.17 Glcd_Set_Font

This function sets the font that will be used with functions: Glcd_Write_Char and Glcd_Write_Text. The arguments passed to the function are:

```
activeFont:     font to be set. Needs to be formatted as an array of char
aFontWidth:     width of the font characters in dots.
aFontHeight:    height of the font characters in dots.
aFontOffs:      number that represents difference between the mikroC Pro
                for PICcharacter set and regular ASCII set (eg. if 'A' is 65
                in ASCII character, and 'A' is 45 in the mikroC Pro for
                PICcharacter set, aFontOffs is 20)
```

List of supported fonts are:

- Font_Glcd_System3 × 5;
- Font_Glcd_System5 × 7;
- Font_Glcd_5 × 7;
- Font_Glcd_Character8 × 7.

```
Example Call:    Glcd_Set_Font(&MyFont, 5, 7, 32);
                 //Use custom 5x7 font MyFont which starts with space
                 character (32)
```

7.3.18 Glcd_Set_Font_Adv

This function sets the font that will be used with functions: Glcd_Write_Char_Adv and Glcd_Write_Text_Adv. The arguments passed to the function are:

```
activeFont:          font to be set. Needs to be formatted as an array of char.
font_colour:         sets font colour.
font_orientation:    sets font orientation.
```

```
Example Call:        Glcd_Set_Font_Adv(&MyFont, 0, 0);
```

www.it-ebooks.info

7.3.19 Glcd_Write_Char

This function displays a character on the GLCD. If no font is specified, then the default Font_Glcd_System5 × 7 font supplied with the library will be used. The arguments passed to the function are:

```
chr:          character to be displayed
x_pos:        character starting position on x-axis (0 to 127- FontWidth)
page_num:     the number of the page on which the character will be
              displayed (0 to 7)
colour:       colour of the character between 0 and 2. 0 is white, 1 is
              black, 2 inverts each dot

Example Call: Glcd_Write_Char('Z',10,2,1);
              //Display character Z at x position 10, inside page 2
```

7.3.20 Glcd_Write_Char_Adv

This function displays a character on the GLCD at co-ordinates (x, y).

```
ch:           character to be displayed.
x:            character position on x-axis.
y:            character position on y-axis.

Example Call: Glcd_Write_Char_Adv('A',20,10,); // Display A at (20,10)
```

7.3.21 Glcd_Write_Text

This function displays text on the GLCD. If no font is specified, then the default Font_Glcd_System5 × 7 font supplied with the library will be used. The arguments passed to the function are:

```
text:         text to be displayed
x_pos:        text starting position on x-axis.
page_num:     the number of the page on which text will be displayed (0 to 7)
colour:       The colour parameter between 0 and 2. 0 is white, 1 is black
              and 2 inverts eachdot.

Example Call: Glcd_Write_Text("My Computer",10,3,1);
              //Display "My Computer" at x position 10 in page 3
```

7.3.22 Glcd_Write_Text_Adv

This function displays text on the GLCD at co-ordinates (x, y). The arguments passed to the function are:

www.it-ebooks.info

```
text:           text to be displayed
x:              text position on x-axis.
y:              text position on y-axis.

Example Call:   Glcd_Write_Text_Adv("My Computer", 10, 10);
                //Display text "My Computer" at coordinates (10,10)
```

7.3.23 *Glcd_Write_Const_Text_Adv*

This function displays text on the GLCD, where the text is assumed to be located in the program memory of the microcontroller. The text is displayed at co-ordinates (x, y). The arguments passed to the function are:

```
text:       text to be displayed
x:          text position on x-axis.
y:          text position on y-axis.

                const char Txt[ ] = "My Computer";
Example Call:   Glcd_Write_Text_Adv(Txt, 10, 10);
                //Display text "My Computer" at coordinates (10,10)
```

7.3.24 *Glcd_Image*

This function displays a bitmap image on the GLCD. The image to be displayed is passed as an argument to the function. The bitmap image array must be located in the program memory of the microcontroller. The GLCD Bitmap Editor of mikroC Pro for PIC compiler can be used to convert an image to a constant, so that it can be displayed by this function.

```
Example Call:       Glcd_Image(MyImage);
```

7.4 Example GLCD Display

An example program is given in this section, to show how the various GLCD library functions can be used. The example program listing is shown in Figure 7.5. Notice that at the beginning of the program, PORT B and PORT D I/O pins are configured as digital by clearing ANSELB and ANSELD registers to 0. This is necessary, since we are using a PIC18F45K22 type microcontroller. Different PIC microcontrollers may require different settings to configure their analogue ports as digital. The images displayed by this program are shown in Figure 7.6. This image was obtained by connecting a GLCD to the EasyPIC 7 development board. The following shapes are drawn on the GLCD:

- a rectangle with rounded edges at co-ordinates (5, 5), (123, 59) and edge radius 10;
- a rectangle at co-ordinates (15, 15), (113, 49);
- a line from (50, 30) to (70, 30);
- a circle with centre at (30, 30) and radius 10;

www.it-ebooks.info

```
/************************************************************
                GLCD LIBRARY FUNCTIONS EXAMPLE
                ------------------------------

This program uses some of the mikroC GLCD library functions to show
how the functions should be used in programs.

The program was loaded to a PIC18F45K22 microcontroller and operated with a
8MHz crystal. The EasyPIC 7 development board is used for this demo

Author:  Dogan Ibrahim
Date:    October, 2011
************************************************************/

// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

void main()
{
   ANSELB = 0;                                    // Configure PORT B as digital I/O
   ANSELD = 0;                                    // Configure PORT D as digital I/O

   Glcd_Init();                                   // Initialize GLCD
   Glcd_Fill(0x00);                               // Clear GLCD
   Glcd_rectangle_round_edges(5,5,123,59,10,1);   // Draw rectangle
   Glcd_Rectangle(15,15,113,49,1);                // Draw rectangle
   Glcd_Line(50, 30, 70, 30, 1);                  // Draw line
   Glcd_Circle(30,30,10,1);                       // Draw circle
   Glcd_Circle_Fill(50,42,5,1);                   // Draw filled circle
   Glcd_Set_Font(Font_Glcd_Character8x7, 8, 7, 32); // Change Font

   Glcd_Write_Text("Txt", 80, 3, 2);              // Write string “Txt”
   Glcd_Write_Text("LCD",80,4,1);                 // Write string “LCD”
   Glcd_Write_Text_Adv("micro",80,38);            // Write string “micro”
}
```

Figure 7.5 Example GLCD program

www.it-ebooks.info

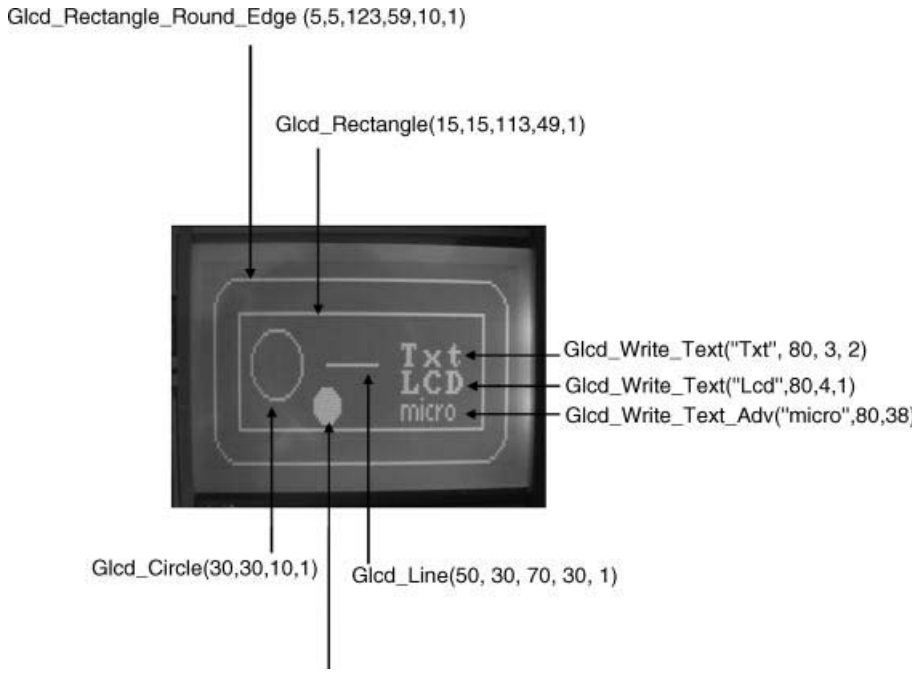


Figure 7.6 Images displayed by the program in Figure 7.5

- a filled circle with centre at (50, 42) and radius 5;
- text 'Txt' at x co-ordinate 80 and page 3;
- text 'LCD' at x co-ordinate 80 and page 4;
- text 'micro' at co-ordinates (80, 38).

The functions used to display each image are also shown in Figure 7.6.

7.5 mikroC Pro for PIC Bitmap Editor

mikroC Pro for PIC provides a tool for converting a bitmap image to a format, which can be displayed on the GLCD screen using the Glcd_Image function described earlier. This tool is accessed by clicking Tools -> GLCD Bitmap Editor from the drop-down menu. The user can choose between three different types of GLCD controllers. You should choose the default KS0108 controller from the tabs at the top of the screen. The GLCD size will automatically be set to 128 × 64 pixels. Choose the compiler as mikroC Pro for PIC from the bottom right-hand corner of the display. Click button *Load BMP* to load the bitmap image you prepared earlier to the program. The image should be displayed on the mikroC Pro for PICGLCD screen. In this example, we have chosen the *Bank* image provided by the compiler as a demo. As shown in Figure 7.7, the mikroC Pro for PIC code to be used to load to the GLCD is automatically created at the bottom of the display.

www.it-ebooks.info

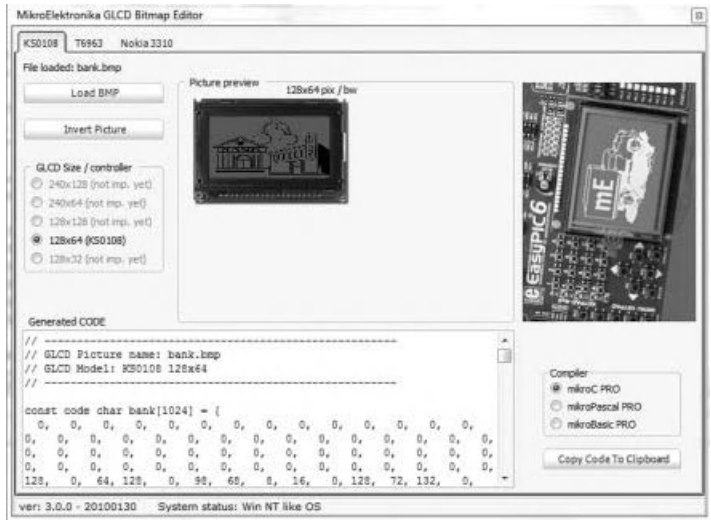


Figure 7.7 ThemikroC Pro for PIC Bitmap Editor

7.6 Adding Touch-screen to GLCDs

More and more consumer electronic products are now available with touch-screen inputs. For example, electronic games, MP3 players, GPS receivers, mobile phones, PDAs, ATM machines, industrial control systems, remote control devices, point-of-sale (POS) terminals, advertisement show screens, information displays, and many more similar products offer special screens where items are selected from a menu by simply touching the relevant part of the screen.

Perhaps the biggest advantage of a touch screen display is that it eliminates the need for keyboard input, resulting in a cheaper and lighter overall design. The user input facilities in such devices are usually provided in the form of *soft* keypads, where the layout of a keypad is displayed on a touch-screen panel, and the required characters and numbers can be entered by simply touching the required key positions on the touch screen. A soft keypad also makes it easier to enter and edit data quickly.

Another advantage of a touch-screen display is that it is usually much quicker to navigate around the screen than using a keyboard or mouse type inputs. Also, in some applications, such as GPS mapping and navigation, a desired geographical point can easily be selected by simply touching the desired point on the screen. It may take more time and effort to accurately select a point on a map using a keyboard and a mouse.

Touch-screen displays are also used in most POS systems, for example in restaurants and supermarket check-outs to select a purchased item from a menu quickly, easily and reliably. Perhaps the main advantage in such applications is the speed of making a correct selection.

Touch-screen displays have some disadvantages. Perhaps one of the greatest is that the screen may get dirty, oily, and finger prints can accumulate on the screen after long usage by the finger and as a result, it may become less sensitive to touch. Also, the screen can easily become scratched, especially if a hard object is used to touch and navigate through the

www.it-ebooks.info

screen. Touch screens can also cause stress on human fingers when used for more than a few minutes at a time, since pressure is required to make a selection. A touch device (e.g. a stylus) or fingernails can be used to prevent issues of direct touch. Another disadvantage is that a touch-screen LCD display is usually more expensive than a standard LCD display. The choice of whether or not to use a touch-screen display depends entirely on the nature of the application, the cost and the level of user experience.

7.6.1 Types of Touch-screen Displays

Touch-screen displays are in the form of either large screen monitors, used for example in PC systems, or small LCDs, used in microcontroller based systems. Although the principle of operation is the same in either case, in this book the small LCD type touch-screen displays are considered. Such displays usually have resolutions of 128×64 pixels and are used in battery operated intelligent devices.

A touch-screen LCD is basically a combination of a GLCD and a touch sensitive panel mounted on top of the GLCD. The two parts are independent of each other: The panel senses the co-ordinates where the user touches the screen, and the GLCD displays graphical information on the LCD display based upon the user's selection.

There are several types of touch-screen LCDs, such as resistive, capacitive, surface acoustic wave, optical imaging, strain gauge, and so on. The most commonly used are the resistive and capacitive types, and some information about each type is given below.

A resistive touch screen consists of several layers, where two electrically conductive resistive layers are separated by a very small gap and a flexible layer is used at the top. One of the layers is connected to a voltage source. When a point is pressed on the screen, the touched points of both conductive layers make a contact and if the voltage is read at the other layer, this voltage will be proportional to the position of the point touched because of the voltage dividing effect. Further details about resistive touch screens are given later.

A capacitive touch screen panel is coated with a material that stores electrical charges. When the panel is touched, a small amount of charge is drawn to the point of contact and the charge is measured at each corner of the panel and is then processed to determine the point touched.

Resistive touch screens have the advantages that the screen responds when touched with any kind of object, for example finger, stylus, nail, and so on. On the other hand, the capacitive screens respond only when touched by a naked finger (e.g. they will not respond when touched with an object or if wearing gloves). On the other hand, capacitive touch screens are lower power devices, have higher granularities, and also provide higher clarity.

Resistive touch screens are used in microcontroller based projects in later chapters in this book. Further information about resistive touch screens is given in the next section.

7.6.2 Resistive Touch Screens

Resistive touch screens are used in most low cost, medium resolution systems. A resistive touch screen consists of at least three layers. As shown in Figure 7.8, the touch screen is mounted on a GLCD. The bottom layer is glass (or acryl), coated with a resistive Indium Tin Oxide (ITO) solution. On top of this, a resistive ITO coated poly Ethylene Terephthalate

www.it-ebooks.info

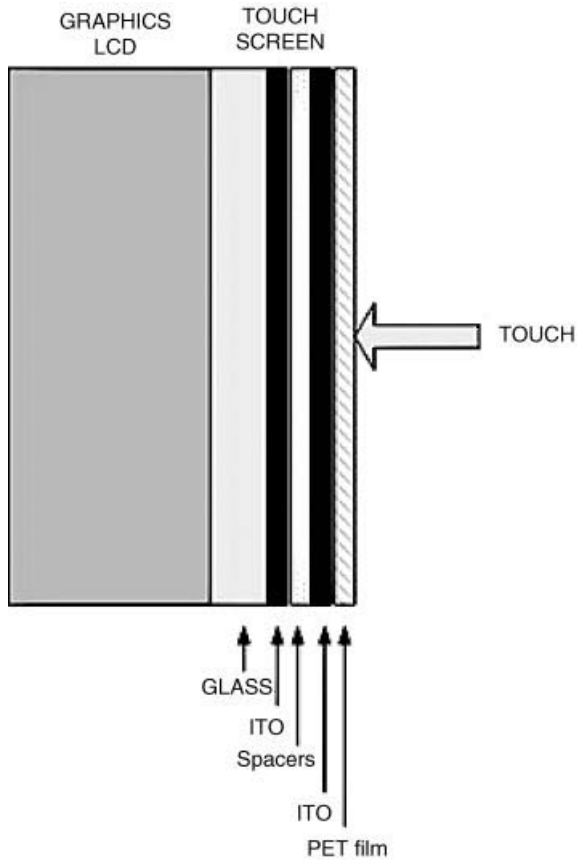


Figure 7.8 Resistive touch screen

(PTE) flexible film is used. The two conductive ITO layers are separated from each other with microdot spacers, so that there is no contact between them when the screen is not touched. When a pressure is applied to the top of the screen, for example by touching the screen, the two ITO layers will make contact at the point of the touch. Electrical circuits are then used to determine the point of the contact. Usually a 4-wire, 5-wire or 8-wire circuit is used to determine the co-ordinates of the point touched by the user. These circuits are described below in greater detail.

7.6.2.1 4-Wire Resistive Touch Screen

These are the least expensive and most commonly used types of resistive touch screens. Conductive bus bars with silver ink are implanted at the opposite edges of a screen layer. The principle of operation is such that, as shown in Figure 7.9, if one side of a layer is connected to +V and the other side to ground, a potential gradient results on the screen layer, and the voltage at any point on this layer becomes directly proportional to the distance from the +V side.

www.it-ebooks.info

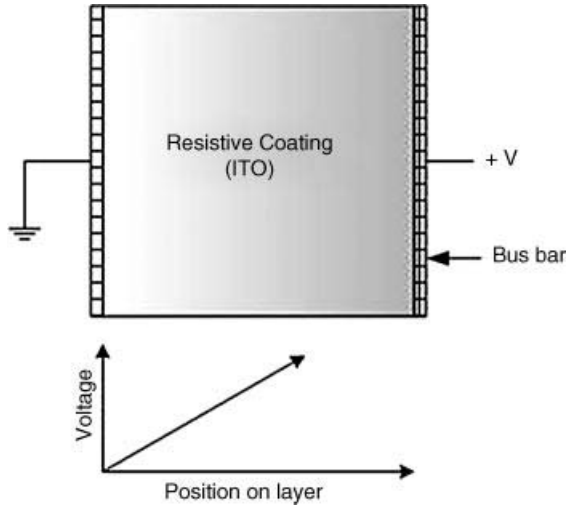


Figure 7.9 Voltage gradient in a screen layer

In a 4-wire touch screen, two measurements are made one after the other, to determine the X and Y co-ordinates of the point touched by the user. Figure 7.10a shows how the X co-ordinate can be determined. Here, the right- and left-hand sides of the top layer can be connected to +V and ground, respectively. The bottom layer can then be used to sense and measure the voltage at the point touched by the user. An A/D converter is used to convert this analogue voltage to digital and then to determine the X co-ordinate.

Similarly, Figure 7.10b shows how the Y co-ordinate can be determined. Here, the upper and lower sides of the bottom layer can be connected to +V and ground, respectively. The top layer can then be used to sense and measure the voltage at the point touched by the user.

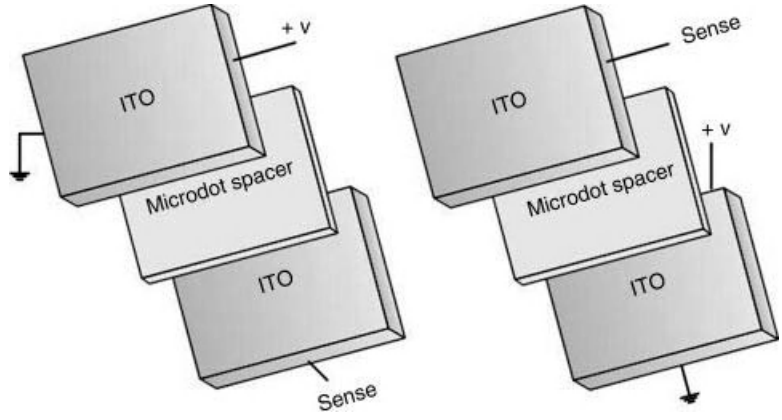


Figure 7.10 Determining the X and Y co-ordinates (4-wire). (a) Determining the X co-ordinate. (b) Determining the Y co-ordinate

www.it-ebooks.info

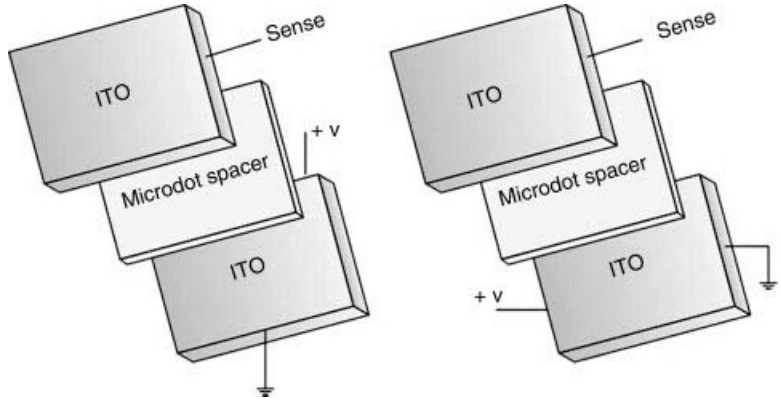


Figure 7.11 Determining the X and Y co-ordinates (5-wire). (a) Determining the X co-ordinate (b) Determining the Y co-ordinate

Again, an A/D converter is used to convert the voltage to digital and then to determine the Y co-ordinate.

7.6.2.2 5-Wire Resistive Touch Screen

This is a modification of the basic 4-wire system, where one layer (usually the top layer) is used for sensing and measuring the voltage, while the other layer is where the voltage gradient is created in X and Y directions. As shown in Figure 7.11a, to determine the X co-ordinate, the upper and lower sides of the bottom layer can be connected to +V and ground, respectively. The top layer is then used to sense and measure the voltage.

To determine the Y co-ordinate, we simply have to reverse polarity and sides of the bottom layer (see Figure 7.11b). The Y co-ordinate is then read from the top layer.

7.6.2.3 8-Wire Resistive Touch Screen

An 8-wire touch screen is used when more accurate measurements of the screen co-ordinates are required. In 4 and 5 wire implementations, the resistance of the bus bars and the connection circuitry usually introduce offset errors in voltage measurements. These offset errors can drift with temperature, humidity and time. 8-wire touch screens compensate for drift by adding 4 additional reference lines, thus enabling the voltage to be measured directly at the touch screen bus bars. 8-wire touch screens are generally more expensive than others and are not covered further in this book.

Example projects using the touch-screen GLCDs will be given in later chapters of this book.

7.7 Summary

This chapter has outlined briefly the different types of GLCDs available to the microcontroller based system designer. The GLCD chosen and used in this chapter was the commonly used industry standard 128 × 64 pixel GLCD, controlled by the popular KS0108 controller.

www.it-ebooks.info

The pin configuration of the GLCD and its connection to a microcontroller has been explained.

mikroC Pro for PIC language supports a large number of library functions for several types of GLCDs. In this chapter, the functions available for the KS0108 type GLCDs have been explained in detail and example function calls are given.

mikroC Pro for PIC language also provides a tool for converting bitmap images to a format that can be loaded to GLCD displays with the help of mikroC Pro for PIC programs. The use of this tool has been explained with the aid of an example.

Finally, operation of the touch-screen GLCDs has been described in detail, with reference to resistive type touch-screen panels.

Exercises

7.1 Give some application areas of GLCDs.
7.2 Draw a circuit diagram to show the connection between a GLCD and a microcontroller. How many I/O pins are required to interface to a GLCD?
7.3 Explain operation of the KS0108 type GLCDs. How many pixels are there? How are the pixels organised?
7.4 Draw a diagram to show the x and y axes of the standard 128×64 pixel GLCD. Mark the (0,0) point on your diagram.
7.5 Show how the GLCD library functions can be used to draw a rectangle.
7.6 Show how the GLCD library functions can be used to draw a circle.
7.7 Explain how a bitmap image that you have created can be loaded and displayed on a GLCD.
7.8 Explain how many types of touch screens are available. What are the advantages and disadvantages of each type?
7.9 Explain how the co-ordinates of a point touched on the screen can be read by a microcontroller.
7.10 What are the advantages and disadvantages of resistive and capacitive touch screens?

www.it-ebooks.info

8

Microcontroller Program Development

Before writing a program, it is always helpful first to drive the program's algorithm. Although simple programs can easily be developed by writing the code prior to any preparation, the development of complex programs almost always become easier if the algorithm is first derived. Once the algorithm is ready, writing of the actual program code is not a difficult task.

A program's algorithm can be described in a variety of graphic and text-based methods, such as a flowchart, structure chart, data flow diagram, program description language (PDL), and so on. The problem with graphical techniques is that it can be very time consuming to draw shapes with text inside them. Also, it is a tedious task to modify an algorithm described using graphical techniques.

Flowcharts can be very useful to describe the flow of control and data in small programs where there are only a handful of diagrams, usually not extending beyond a page or two. The PDL can be useful to describe the flow of control and data in small to medium size programs. The main advantage of the PDL description is that it is very easy to modify a given PDL, since it only consists of text.

In this chapter, we will mainly be using the PDL, but flowcharts will also be given where it is felt to be useful. The next sections briefly describe the basic building blocks of the PDL and flowcharts. It is left to the reader to decide which method to use during the development of their programs.

8.1 Using the Program Description Language and Flowcharts

Program description language (PDL) is free-format English-like text, which describes the flow of control and data in a program. PDL is not a programming language, but a collection of keywords that enable a programmer to describe the operation of a program in a stepwise and logical manner. In this section we will look at the basic PDL statements and their flowchart equivalents. The superiority of the PDL over flowcharts will become obvious when we have to develop medium to large programs.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

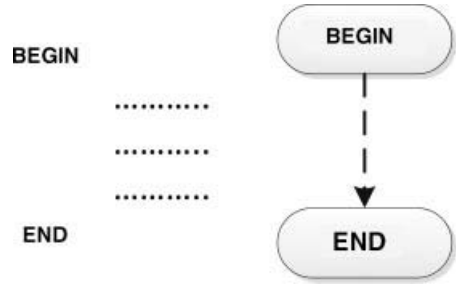


Figure 8.1 BEGIN – END statement and equivalent flowchart

8.1.1 BEGIN – END

Every PDL program description should start with a BEGIN and end with an END statement. The keywords in a PDL description should be highlighted to make the reading easier. The program statements should be indented and described between the PDL keywords. An example is shown in Figure 8.1, together with the equivalent flow diagram.

8.1.2 Sequencing

For normal sequencing, the program statements should be written in English text and describe the operations performed. An example is shown in Figure 8.2, together with the equivalent flowchart.

8.1.3 IF – THEN – ELSE – ENDIF

The IF, THEN, ELSE and ENDIF should be used to conditionally change the flow of control in a program. Every IF line should be terminated with a THEN, and every IF block should be

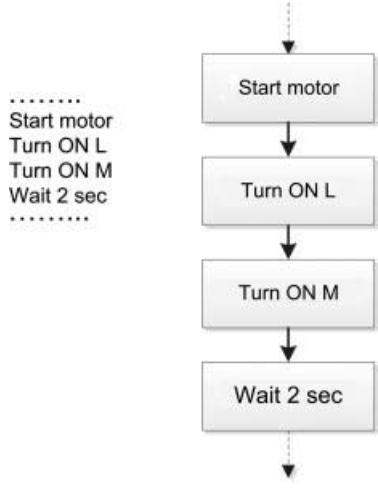


Figure 8.2 Sequencing and equivalent flowchart

www.it-ebooks.info

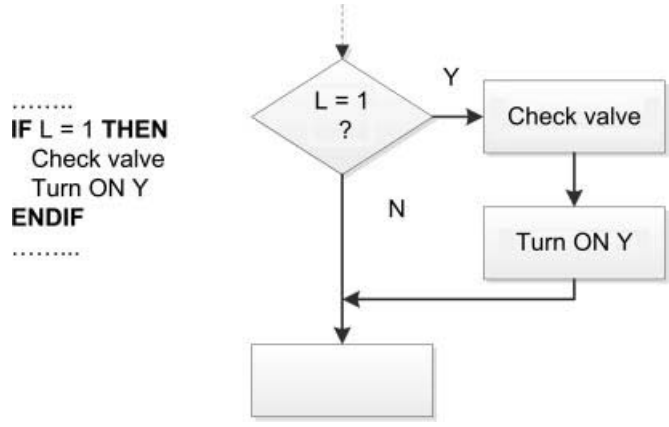


Figure 8.3 Using IF – THEN – ENDIF statements

terminated with an ENDIF statement. Use of the ELSE statement is optional and depends on the application. Figure 8.3 shows an example of using IF – THEN – ENDIF, while Figure 8.4 shows the use of IF – THEN – ELSE – ENDIF statements in a program and their equivalent flowcharts.

8.1.4 DO – ENDDO

The DO – ENDDO statements should be used when it is required to create iterations, or conditional or unconditional loops in programs. Every DO statement should be terminated with an ENDDO. Other keywords, such as FOREVER or WHILE can be used after

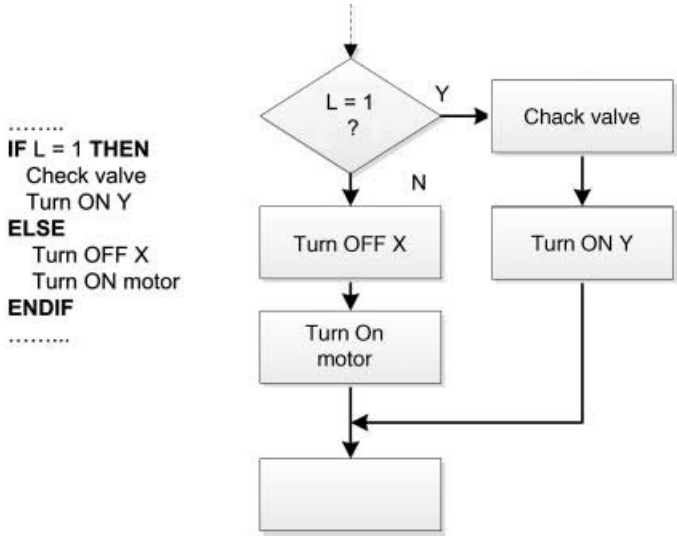


Figure 8.4 Using IF – THEN – ELSE – ENDIF statements

www.it-ebooks.info

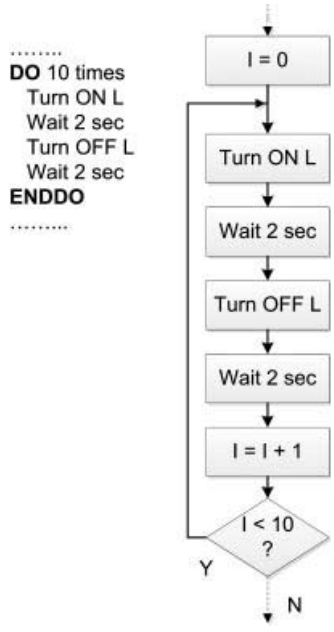


Figure 8.5 Using DO – ENDDO statements

the DO statement, to indicate an endless loop or a conditional loop, respectively. Figure 8.5 shows an example of a DO – ENDDO loop executed 10 times. Figure 8.6 shows an endless loop created using the FOREVER statement. The flowchart equivalents are also shown in the figures.

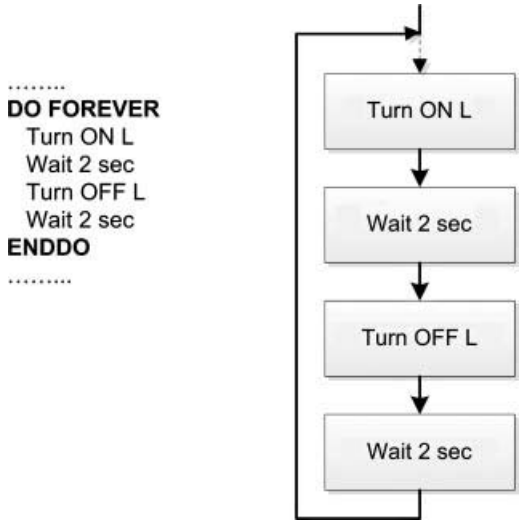


Figure 8.6 Using DO – FOREVER statements

www.it-ebooks.info

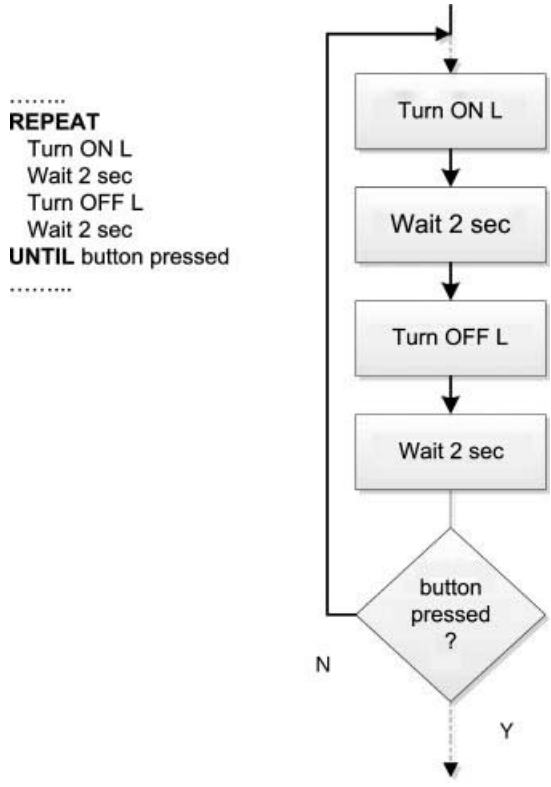


Figure 8.7 Using REPEAT – UNTIL statements

8.1.5 REPEAT – UNTIL

REPEAT – UNTIL is similar to DO – WHILE, but here the statements enclosed by the REPEAT – UNTIL block are executed at least once, while the statements enclosed by DO – WHILE may not execute at all if the condition is not satisfied just before entering the DO statement. An example is shown in Figure 8.7, with the equivalent flowchart.

8.1.6 Calling Subprograms

In some applications, a program consists of a main program and a number of subprograms (or functions). A subprogram activation in PDL should be shown by adding the CALL statement before the name of the subprogram. In flowcharts, a rectangle with vertical lines at each side should be used to indicate the invocation of a subprogram. An example call to a subprogram is shown in Figure 8.8, for both a PDL description and a flowchart.

8.1.7 Subprogram Structure

A subprogram should begin and end with the keywords BEGIN/name and END/name, respectively, where *name* is the name of the subprogram. In flowchart representation, a

www.it-ebooks.info

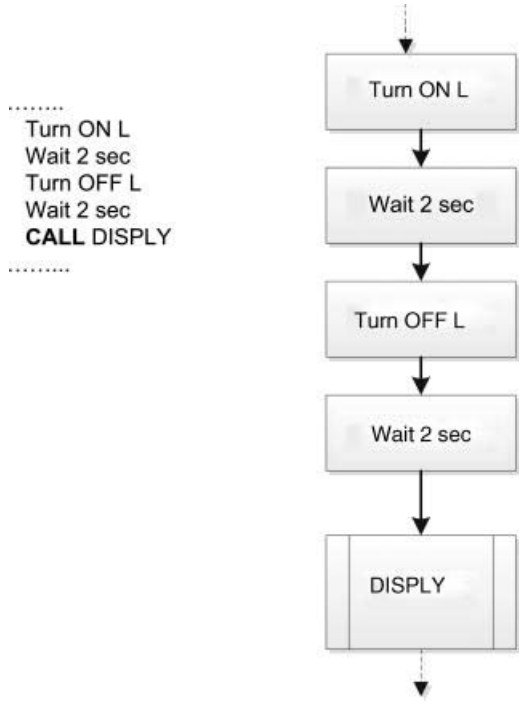


Figure 8.8 Calling a subprogram

horizontal line should be drawn inside the BEGIN box and the name of the subprogram should be written at the lower half of the box. An example subprogram structure is shown in Figure 8.9, for both a PDL description and a flowchart.

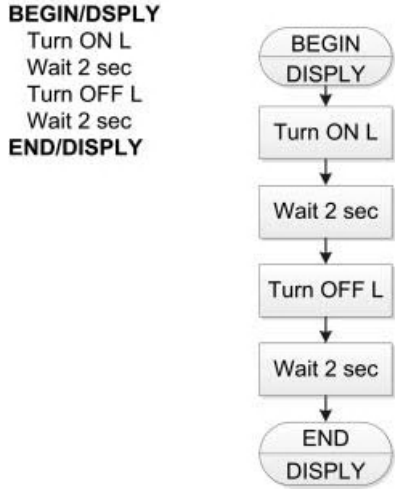


Figure 8.9 Subprogram structure

www.it-ebooks.info

8.2 Examples

Some examples are given in this section, to show how the PDL and flowcharts can be used in program development.

Example 8.1

It is required to a write a program to convert hexadecimal numbers 'A' to 'F' into decimal. Show the algorithm using a PDL and also draw the flowchart. Assume that the number to be converted is called HEX_NUM and the output number is called DEC_NUM.

Solution 8.1
The required PDL is:

```
BEGIN
    IF HEX_NUM = "A" THEN
        DEC_NUM = 10
    ELSE IF HEX_NUM = "B" THEN
        DEC_NUM = 11
    ELSE IF HEX_NUM = "C" THEN
        DEC_NUM = 12
    ELSE IF HEX_NUM = "D" THEN
        DEC_NUM = 13
    ELSE IF HEX_NUM = "E" THEN
        DEC_NUM = 14
    ELSE IF HEX_NUM = "F" THEN
        DEC_NUM = 15
    ENDIF
END
```

The required flowchart is shown in Figure 8.10. Notice that it is much easier to write the PDL statements than drawing the flowchart shapes and writing text inside them.

Example 8.2

The PDL of part of a program is given as follows:

```
J = 0
M = 0
DO WHILE J < 10
    DO WHILE M < 20
        Flash the LED
        Increment M
    ENDDO
    Increment J
ENDDO
```

Show how this PDL can be implemented by drawing a flowchart.

Solution 8.2
The required flowchart is shown in Figure 8.11. Here again, notice how complicated the flowchart can be, even for a simple nested DO WHILE loop.

www.it-ebooks.info

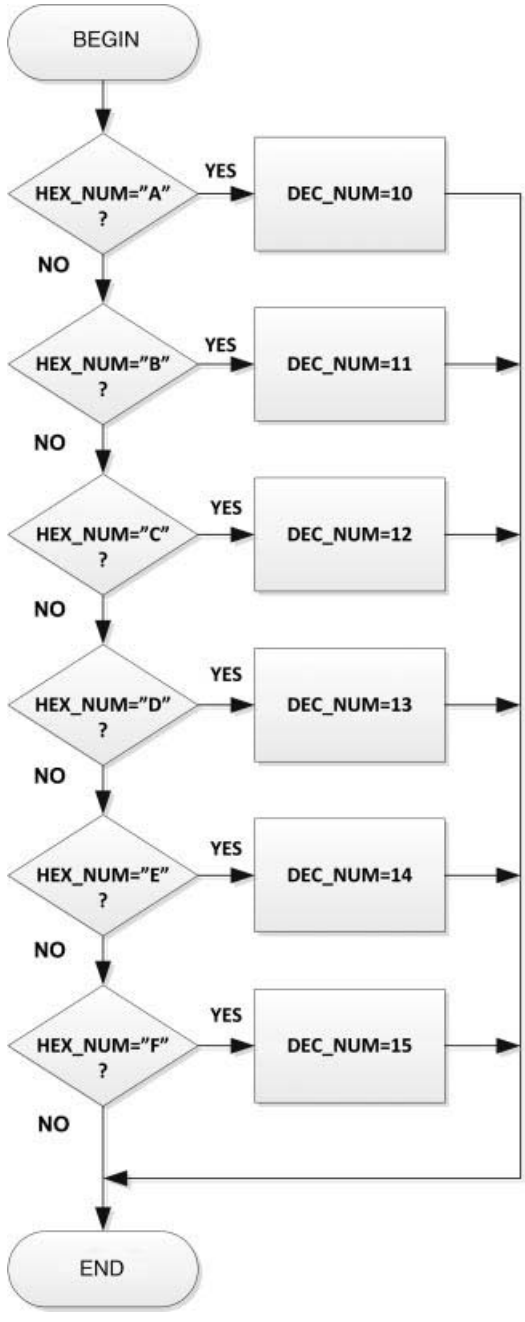


Figure 8.10 Flowchart solution

www.it-ebooks.info

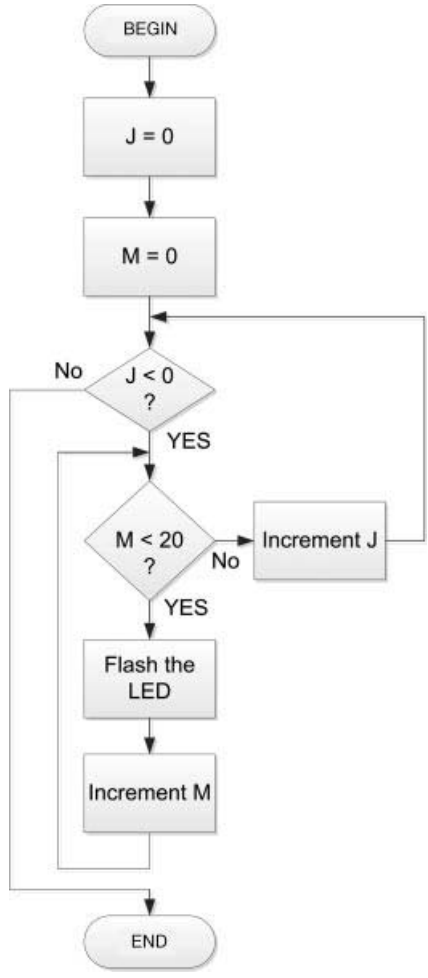


Figure 8.11 Flowchart solution

Example 8.3

It is required to write a program to calculate the sum of integer numbers between 1 and 100. Show the algorithm using a PDL and also draw the flowchart. Assume that the sum will be stored in a variable called SUM.

Solution 8.3
The required PDL is:

```
BEGIN
    SUM = 0
    I = 1
    DO 100 TIMES
```

www.it-ebooks.info

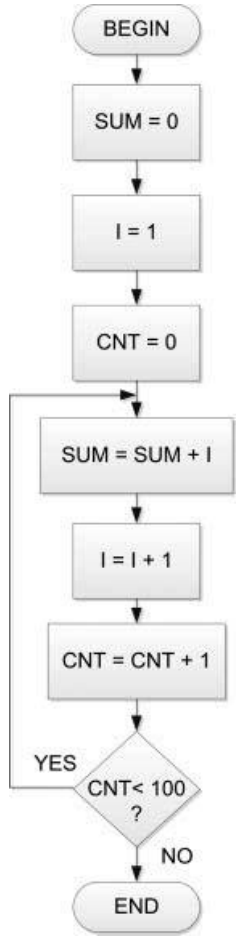


Figure 8.12 Flowchart solution

```
        SUM = SUM + I
        Increment I
    ENDDO
END
```

The required flowchart is shown in Figure 8.12.

Example 8.4

It is required to write a program to calculate the sum of all the even numbers between 1 and 10 inclusive. Show the algorithm using a PDL and also draw the flowchart. Assume that the sum will be stored in a variable called SUM.

www.it-ebooks.info

Solution 8.4

The required PDL is:

```
BEGIN
    SUM = 0;
    CNT = 1
    REPEAT
        IF CNT is even number THEN
            SUM = SUM + CNT
        ENDIF
        INCREMENT CNT
    UNTIL CNT > 10
END
```

The required flowchart is shown in Figure 8.13. Notice how complicated the flowchart can be for a very simple problem such as this.

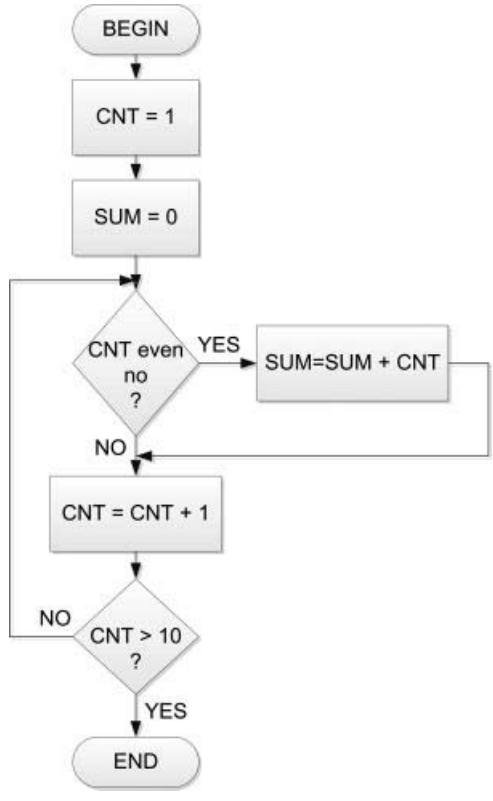


Figure 8.13 Flowchart solution

www.it-ebooks.info

8.3 Representing for Loops in Flowcharts

Most programs include some form of iteration or looping. One of the easiest ways to create a loop in a C program is by using the *for* statement. This section shows how a *for* loop can be represented in a flowchart. As shown below, there are several methods of representing a *for* loop in a flowchart.

Suppose that we have a *for* loop as below and we wish to draw an equivalent flowchart.

```
for(m = 0; m < 10; m++)
{
    Cnt = Cnt + 2*m;
}
```

Method 1

Figure 8.14 shows one of the methods for representing the above *for* loop as with a flowchart. Here, the flowchart is drawn using the basic primitive components.

Method 2

Figure 8.15 shows the second method for representing the *for* loop with a flowchart. Here, a hexagon shaped flowchart symbol is used to represent the *for* loop and the complete *for* loop statement is written inside this symbol.

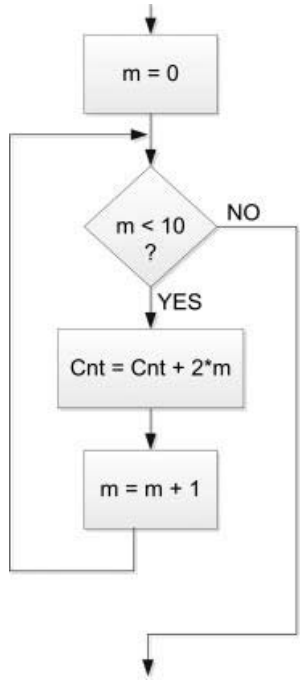


Figure 8.14 Method 1 for representing a *for* loop

www.it-ebooks.info

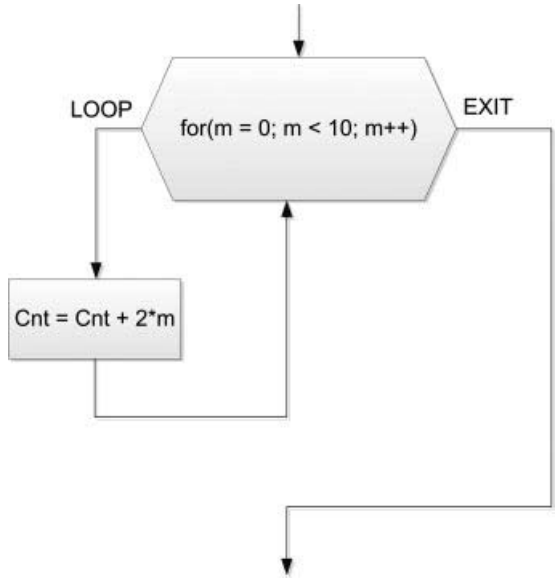


Figure 8.15 Method 2 for representing a *for* loop

Method 3

Figure 8.16 shows the third method for representing the *for* loop with a flowchart. Here again, a hexagon shaped flowchart symbol is used to represent the *for* loop and the symbol is divided into three to represent the initial condition, the increment, and the terminating condition.

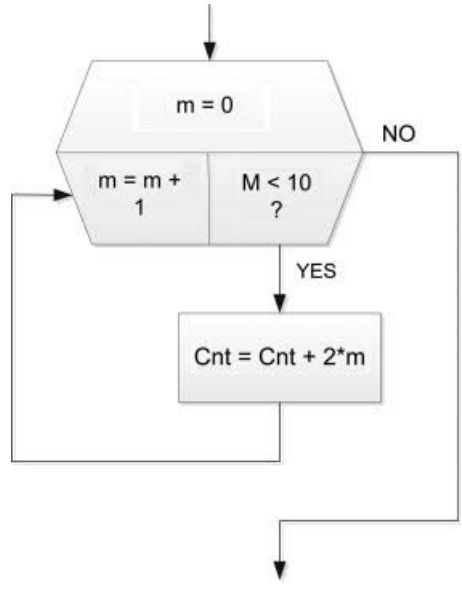


Figure 8.16 Method 3 for representing a *for* loop

www.it-ebooks.info

8.4 Summary

This chapter has described the program development process using the PDL and flowcharts as tools. The PDL is commonly used, as it is a simple and convenient method of describing the operation of a program. The PDL consists of several English-like keywords. Although the flowchart is also a useful tool, it can be very tedious in large programs to draw shapes and write text inside them.

Exercises

8.1 Describe the various shapes used in drawing flowcharts.
8.2 Describe how the various keywords used in PDL can be used to describe the operation of a program.
8.3 What are the advantages and disadvantages of flowcharts?
8.4 It is required to write a program to calculate the sum of numbers from 1 to 10. Draw a flowchart to show the algorithm for this program.
8.5 Write the PDL statements for the question in (4) above.
8.6 It is required to write a program to calculate the roots of a quadratic equation, given the coefficients. Draw a flowchart to show the algorithm for this program.
8.7 Write the PDL statements for the question in (6).
8.8 Draw the equivalent flowchart for the following PDL statements:

```
DO WHILE count < 10
    Increment J
    Increment count
ENDDO
```

8.9 It is required to write a function to calculate the sum of numbers from 1 to 10. Draw a flowchart to show how the function subprogram and the main program can be implemented.
8.10 Write the PDL statements for the question (9) above.
8.11 It is required to write a function to calculate the cube of a given integer number and then call this function from a main program. Draw a flowchart to show how the function subprogram and the main program can be implemented.
8.12 Write the PDL statements for the question (8) above.
8.13 Draw the equivalent flowchart for the following PDL statements:

```
J = 0
K = 0
REPEAT
    Flash LED A
    Increment J
    REPEAT
        Flash LED B
        Increment K
    UNTIL K = 10
UNTIL J > 15
```

www.it-ebooks.info

9

LED Based Projects

In this and subsequent chapters, we will look at the design of projects using display devices, such as LEDs, LCDs and GLCDs. The EasyPIC 7 development board is used in most of the projects, but the full circuit diagram is given for the readers who may want to build a project, but who may not have the EasyPIC 7 development board. We will start with very simple projects and increase the complexity. All the projects given in the book are fully tested and working. It is recommended that the reader moves through the projects in the given order, to gain the maximum benefit.

The following will be provided for each project:

- the title;
- brief description;
- block diagram;
- circuit diagram;
- algorithmic description (PDL or flowchart);
- program listing;
- program description;
- suggestions for further development.

9.1 PROJECT 9.1 – Flashing LED

9.1.1 Project Description

This is perhaps the simplest project we can have. In this project, an LED is connected to bit 0 of PORT C (RC0) of a PIC18F45K22 type microcontroller (other PIC microcontrollers can also be used). The reason for using this microcontroller is because the EasyPIC 7 development board is equipped with this type of PIC microcontroller), operated from an 8 MHz crystal. The LED is flashed with 1 second intervals.

As described in Section 5.1, an LED is connected to a microcontroller using a current limiting resistor. An LED can be connected in two modes to a microcontroller output port: current sinking mode or current sourcing mode.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

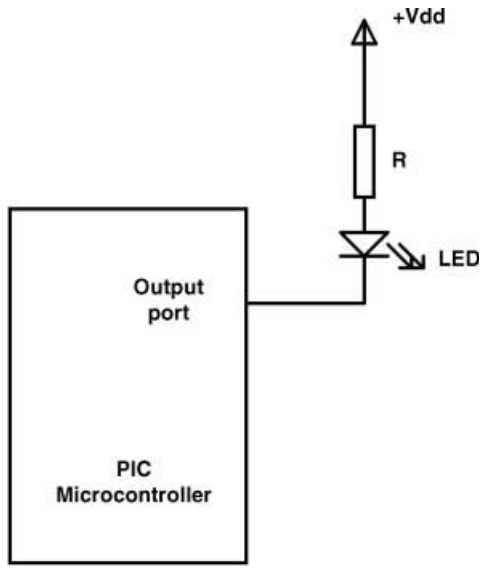


Figure 9.1 LED connected in current sinking mode

9.1.1.1 Current Sinking Mode

As shown in Figure 9.1, in current sinking mode the cathode of the LED is connected to the microcontroller I/O port, while the anode is connected to +5 V supply through a current limiting resistor. Here, the LED is turned ON when the output of the microcontroller is at logic LOW, where current flows into the microcontroller pin. The output current sink capability of each PIC microcontroller I/O pin is 25 mA. As was shown in Section 5.1, a 330 ohm or smaller resistor should give a current of about 10 mA, which is sufficient to drive the LED to give bright light.

9.1.1.2 Current Sourcing Mode

As shown in Figure 9.2, in current sourcing mode the anode of the LED is connected to the microcontroller I/O port, while the cathode is connected to the ground through a current limiting resistor. Here, the LED is turned ON when the output of the microcontroller is at logic HIGH, where current flows out of the microcontroller pin. The output current source capability of each PIC microcontroller I/O pin is 25 mA. As in the current sinking mode, a 330 ohm or smaller resistor should provide the required LED brightness.

9.1.2 Block Diagram

The block diagram of the project is shown in Figure 9.3.

www.it-ebooks.info

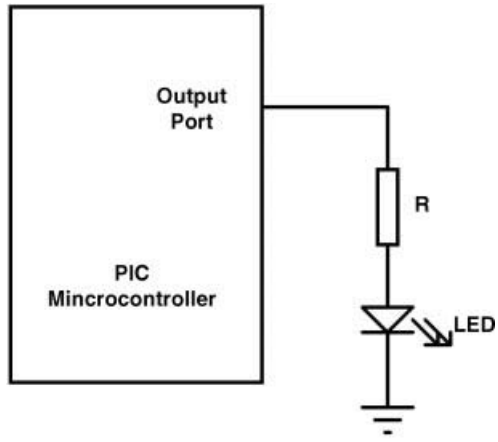


Figure 9.2 LED connected in current sourcing mode

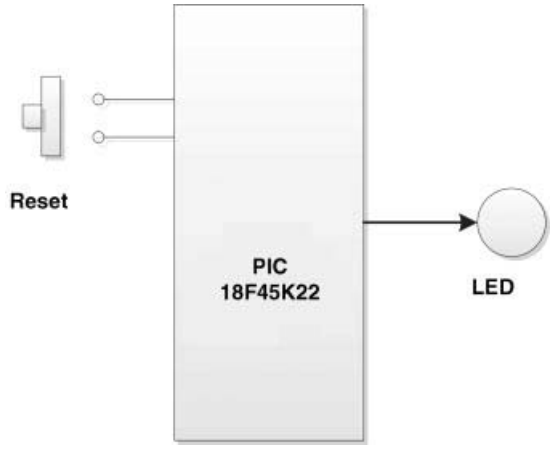


Figure 9.3 Block diagram of the project

9.1.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 9.4. If using the EasyPIC 7 development board, there is no need to change any jumper settings. An 8 MHz crystal is used to provide the clock signal. The microcontroller is Reset using an external push-button switch.

9.1.4 Project PDL

The PDL of this project is very simple and is given in Figure 9.5.

www.it-ebooks.info

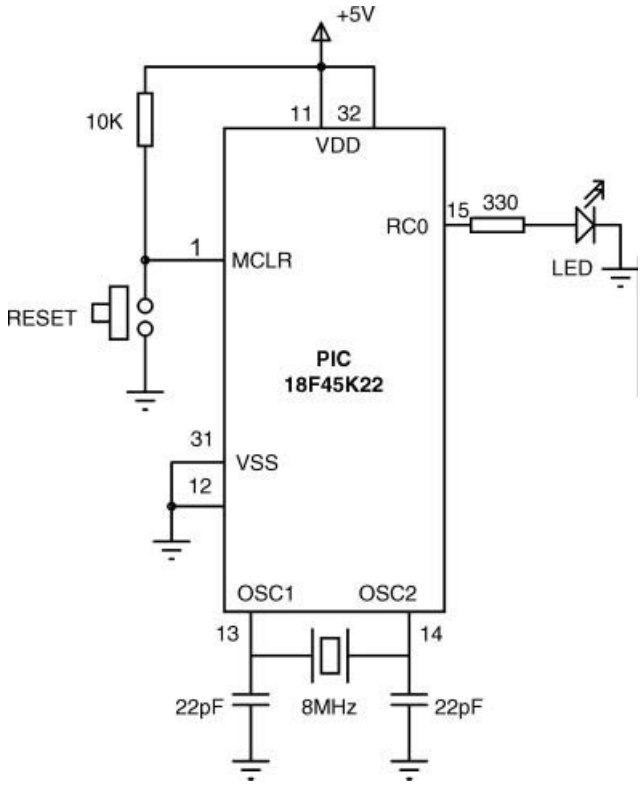


Figure 9.4 Circuit diagram of the project

9.1.5 Project Program

The program is named LED1.C and the program listing of the project is shown in Figure 9.6. At the beginning of the project, PORT C is configured as a digital I/O port by clearing register ANSELC (different PIC microcontrollers may require different settings). Then all the I/O ports of PORT C are set to be outputs by clearing the TRISC register. An endless *for* loop is then constructed and the LED is flashed with 1 second intervals using the *Delay_Ms* function

```
BEGIN
        Configure PORT C
        DO FOREVER
                Turn ON LED
                Wait 1 second
                Turn OFF LED
                Wait 1 second
        ENDDO
END
```

Figure 9.5 PDL of the project

www.it-ebooks.info

```
/*********************************************************************************************
                                    FLASHING LED
                                    ------------

In this project an LED is connected to bit 0 of PORT C (RC0) of a PIC18F45K22 type
microcontroller (any other PIC microcontroller can also be used in this project). The
microcontroller is operated from an 8MHz crystal. The LED is flashed continuously
with 1 second intervals.

Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED1.C
*********************************************************************************************/

void main()
{
  ANSELC = 0;                          // Configure PORT C as digital
  TRISC = 0;                           // Configure PORT C as outputs

  for(;;)                              // FOREVER loop
  {
     PORTC.RC0 = 1;                    // Turn ON LED
     Delay_Ms(1000);                   // Wait 1 second
     PORTC.RC0 = 0;                    // Turn OFF LED
     Delay_Ms(1000);                   // Delay 1 second
  }
}
```

Figure 9.6 Program listing

with an argument of 1000 (i.e. 1 s). Notice that the individual bits of a port can be accessed as PORTn.Rnb, where n is the port name (A, B, C, D, etc.) and b is the bit number (0 to 7), or as PORTn.Fb, or as Rnb_bit. Thus, bit 0 of PORT C can either be accessed as PORTC.RC0, or as PORTC.F0, or as RC0_bit.

The program given in Figure 9.6 can be made more user friendly and easier to read if '#define' pre-processor statements are used, as shown in program LED2.C in Figure 9.7.

9.1.6 Suggestion for a Change

The program given in Figure 9.6 simply flashes the LED with 1 second intervals. This program can be modified, for example to simulate the flashing of lighthouse lights for maritime educational purposes. Different lighthouse lights have different flashing characteristics (see http://en.wikipedia.org/wiki/Light_characteristic), such as alternating, fixed, flashing, occulting, quick flash, very quick flash, and so on.

As an example, we shall modify the program to simulate the lighthouse signal known as

```
VQ(3) 5s
```

This signal consists of 3 short flashes, each 500 ms on and 100 ms off, and repeated every 5 seconds (see Figure 9.8). The PDL of the modified program is shown in Figure 9.9, and the

www.it-ebooks.info

```
/*********************************************************************************************************
                                        FLASHING LED
                                        ------------

In this project an LED is connected to bit 0 of PORT C (RC0) of a PIC18F45K22 type
microcontroller (any other PIC microcontroller can also be used in this project).
The microcontroller is operated from an 8MHz crystal. The LED is flashed continuously
with 1 second intervals.

In this version of the program "define" pre-processor statements are used to make the
program more readable.

Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED2.C
*********************************************************************************/
#define LED PORTC.F0                        // LED is bit 0 of PORT C
#define ON 1
#define OFF 0
#define Delay_1_second Delay_Ms(1000)

void main()
{
   ANSELC = 0;                              // Configure PORT C as digital
   TRISC = 0;                               // Configure PORT C as outputs

   for(;;)                                  // FOREVER loop
   {
      LED = ON;                             // Turn ON LED
      Delay_1_second;                       // Wait 1 second
      LED = OFF;                            // Turn OFF LED
      Delay_1_second;                       // Delay 1 second
   }
}
```

Figure 9.7 More user friendly program

program (named LED3.C) listing is shown in Figure 9.10. Notice that the period of the signal is 5 seconds. The 3 flashes take 1.7 seconds, leaving 3.3 seconds before the flashing starts again. The program contains a nested *for* loop. where the inner loop is repeated 3 times and the outer loop is repeated forever.

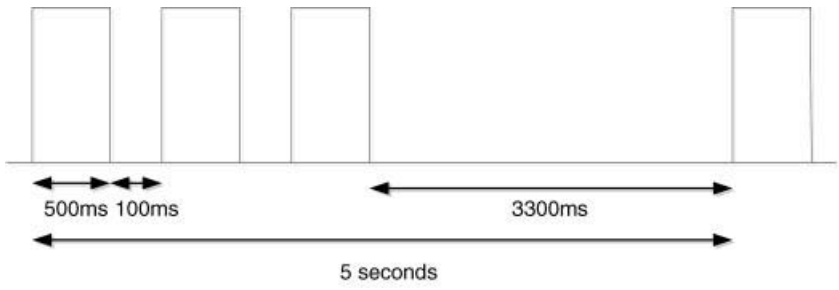


Figure 9.8 VQ(3) 10 s lighthouse signal

www.it-ebooks.info

```
BEGIN
        Configure PORT C
        DO FOREVER
                DO 3 times
                        Turn ON LED
                        Wait 500ms
                        Turn OFF LED
                        Wait 100ms
                ENDDO
                Wait 3.3 seconds
        ENDDO
END
```

Figure 9.9 PDL of the modified program

```
/*************************************************************************************************
                        FLASHING LIGHTHOUSE LED
                        -----------------------------------------

In this project an LED is connected to bit 0 of PORT C (RC0) of a PIC18F45K22 type
microcontroller (any other PIC microcontroller can also be used in this project).
The microcontroller is operated from an 8MHz crystal.

In this project the LED simulates the flashing of a lighthouse light having the
characteristics:

 VQ(3) 5s

where the light flashes 3 times with 500ms ON time and 100ms OFF time with a period
of 5 seconds.

Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED3.C
*************************************************************************************************/

void main()
{
  unsigned char i;
  ANSELC = 0;                           // Configure PORT C as digital
  TRISC = 0;                            // Configure PORT C as outputs

  for(;;)                               // FOREVER loop
  {
    for(i= 0; i < 3; i++)               // Do 3 times
    {
        PORTC.RC0 = 1;                  // Turn ON LED
        Delay_Ms(500);                  // Wait 500 milliseconds
        PORTC.RC0 = 0;                  // Turn OFF LED
        Delay_Ms(100);                  // Wait 100 milliseconds
    }
    Delay_Ms(3300);                     // Wait 3.3 seconds
  }
}
```

Figure 9.10 Listing of the modified program

www.it-ebooks.info

9.2 PROJECT 9.2 – Binary Counting Up LEDs

9.2.1 Project Description

This is a simple project where 8 LEDs are connected to PORT C of a PIC microcontroller. The LEDs count up in binary with a 1 second interval between each count. The pattern displayed by the LEDs will be as shown in Figure 9.11.

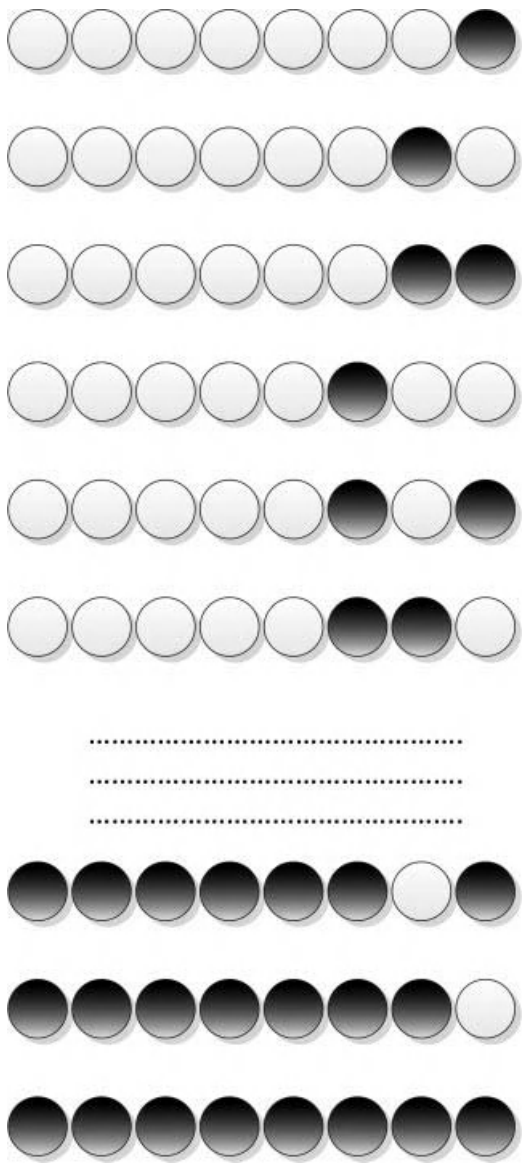


Figure 9.11 Binary counting up LEDs

www.it-ebooks.info

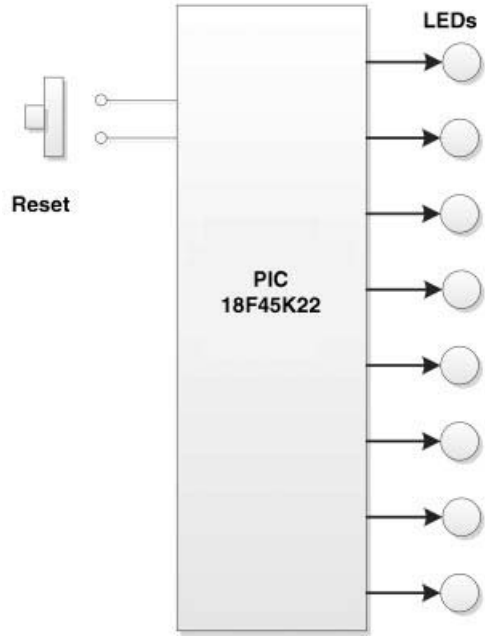


Figure 9.12 Block diagram of the project

9.2.2 *Block Diagram*

The block diagram of the project is shown in Figure 9.12.

9.2.3 *Circuit Diagram*

The circuit diagram of the project is shown in Figure 9.13. If using the EasyPIC 7 development board, there is no need to change any jumper settings. An 8 MHz crystal is used to provide the clock signal. The microcontroller is Reset, using an external push-button switch.

9.2.4 *Project PDL*

The PDL of this project is very simple and is given in Figure 9.14.

9.2.5 *Project Program*

The program is named LED4.C and the program listing of the project is given in Figure 9.15. At the beginning of the project, variable *Cnt* is declared and initialised to 0. Notice that all the variables used in a C program must be declared at the beginning of the program. Then, PORT C is configured as a digital I/O port as before and an endless *for* loop is formed. Inside

www.it-ebooks.info

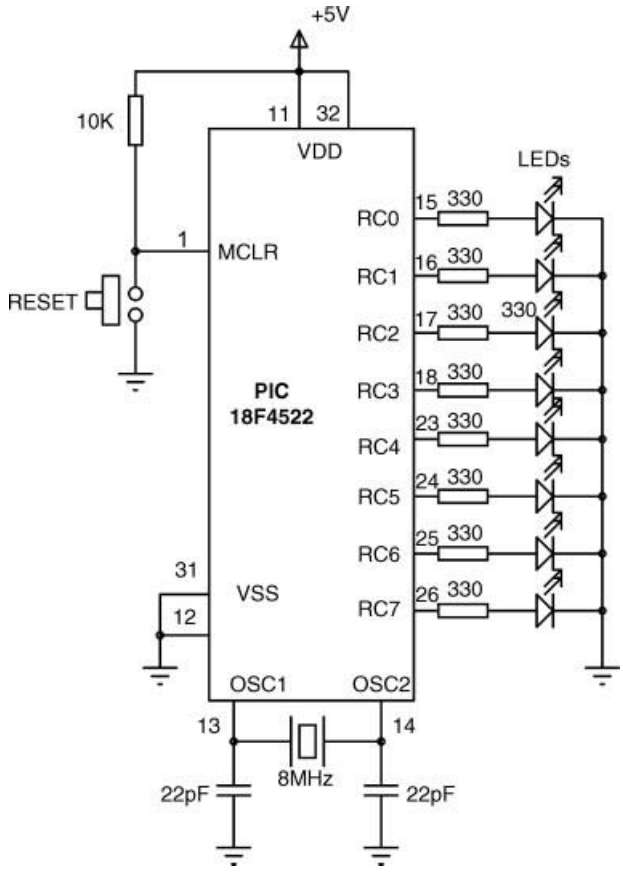


Figure 9.13 Circuit diagram of the project

this loop, variable *Cnt* is sent to PORT C, then after a delay of 1 second *Cnt* is incremented and the loop is repeated forever.

9.2.6 *Suggestions for Further Development*

As an example, the project can be modified to count down after it reaches 255, or to count in steps of an integer number, for example in steps of 2.

```
BEGIN
        Configure PORT C
        Cnt = 0
        DO FOREVER
                Send Cnt to PORT C
                Wait 1 second
                Increment Cnt
        ENDDO
END
```

Figure 9.14 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************************
                              BINARY COUNTING UP LEDs
                              -----------------------

In this project 8 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller
(any other PIC microcontroller can also be used in this project). The microcontroller is
operated from an 8MHz crystal.

In this project the LEDs count up in binary from 0 to 255 and then back to 0 with one
second delay between each count.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED4.C
*************************************************************************************************/

void main()
{
  unsigned char Cnt = 0;                  // Declare and initialise Cnt
  ANSELC = 0;                             // Configure PORT C as digital
  TRISC = 0;                              // Configure PORT C as outputs

  for(;;)                                 // FOREVER loop
  {
     PORTC = Cnt;                         // SendCnt to PORT C
     Delay_Ms(1000);                      // Delay 1 second
     Cnt++;                               // Increment Cnt
  }
}
```

Figure 9.15 Program listing

9.3 PROJECT 9.3 – Rotating LEDs

9.3.1 Project Description

In this project, 8 LEDs are connected to PORT C of a PIC microcontroller as in Project 2. The LEDs turn ON right to left (bit 0 towards bit 7) in a rotating manner, with a 1 second delay between each output where only one LED is ON at any time. If the LEDs are arranged in a circular way, the pattern displayed by the LEDs will be as shows in Figure 9.16.

9.3.2 Block Diagram

The block diagram of the project is as shown in Figure 9.12.

9.3.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.13.

www.it-ebooks.info

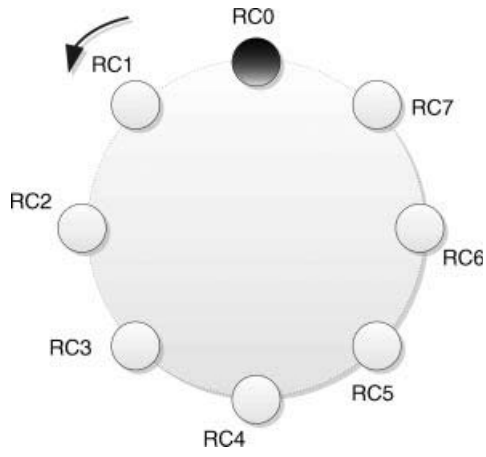


Figure 9.16 Rotating LEDs

9.3.4 Project PDL

The PDL of this project is very simple and is given in Figure 9.17.

9.3.5 Project Program

The program is named LED5.C and the program listing of the project is given in Figure 9.18. At the beginning of the project, variable *Cnt* is initialised to 1. PORT C is configured as a digital I/O port as before. An endless *for* loop is then constructed and *Cnt* is sent to PORT C. After a 1 second delay, *Cnt* is shifted left and the loop is repeated forever. Thus, *Cnt* takes the values 1 2 4 8 16 32 64 128 1 2 . . .

9.3.6 Modification of the Program

The program given in Figure 9.18 rotates the LEDs left. Now we will modify the program so that the LEDs rotate both left and right. In the modified program, after the last LED is lit while rotating in one direction, the direction of rotation will change. Thus, for example,

```
BEGIN
        Cnt = 1
        Configure PORT C
        DO FOREVER
                Send Cnt to PORT C
                Wait 1 second
                Shift left Cnt
                IF Cnt = 0 THEN
                        Cnt = 1
                ENDIF
        ENDDO
END
```

Figure 9.17 PDL of the project

www.it-ebooks.info

```
/*****************************************************************************************************
                                    ROTATING LEDs
                                    -------------------------

In this project 8 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller
(any other PIC microcontroller can also be used in this project).  The microcontroller is
operated from an 8MHz crystal.

In this project the LEDs rotate left (RB0 through RB1 and so on) with one second delay
between each output.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED5.C
*****************************************************************************************************/

void main()
{
  unsigned char Cnt = 1;                    // Initialise Cnt
  ANSELC = 0;                               // Configure PORT C as digital
  TRISC = 0;                                // Configure PORT C as outputs

  for(;;)                                   // FOREVER loop
  {
     PORTC = Cnt;                           // Send Cnt to PORT C
     Delay_Ms(1000);                        // Delay 1 second
     Cnt = Cnt << 1;                        // Shift left 1 digit
     if(Cnt == 0)Cnt = 1;                   // If the last LED re-start
  }
}
```

Figure 9.18 Program listing

while rotating right to left, if the LED at position RC7 is lit, then the rotation direction will change to be from left to right, and the LED at position RC6 will be lit next, and so on. The PDL of the modified program is shown in Figure 9.19.

The listing of the modified program (LED6.C) is shown in Figure 9.20. As before, a variable called *Cnt* is initialised and PORT C is configured as digital I/O. In addition, a variable called *Mode* is used, which determines the direction of rotation. When *Mode* is 0, the LEDs rotate from right to left, and when *Mode* is 1, the LEDs rotate from left to right. When the last LED in a row is lit, the direction of rotation is changed by setting a new value in *Cnt* and changing the value of *Mode*. Thus, *Cnt* takes the values 1 2 4 8 16 32 64 128 64 32 16 8 4 2 1 2 . . .

9.4 PROJECT 9.4 – Wheel of Lucky Day

9.4.1 Project Description

In this project, 7 LEDs are connected to PORT C of a PIC microcontroller. In addition, a push-button switch (START) is connected to bit 7 of PORT C. The LEDs are numbered with

www.it-ebooks.info

```
BEGIN
        Cnt = 1
        Mode = 0
        Configure PORT C
        DO FOREVER
                Send Cnt to PORT C
                Wait 1 second
                IF Mode = 0 THEN
                        Left shift Cnt
                        IF Cnt = 0 THEN
                                Cnt = 64
                                Mode = 1
                        ENDIF
                ELSE
                        Right shift Cnt
                        IF Cnt = 0 THEN
                                Cnt = 2
                                Mode = 0
                        ENDIF
                ENDIF
        ENDDO
END
```

Figure 9.19 PDL of the modified program

the days of the week, as shown in Figure 9.21. When the project is started, the LEDs are turned ON and OFF very fast in a rotating manner, such that it is not possible to see which LED is ON at any time. Pressing the button stops the rotation and only the last LED, which was ON at the time, remains lit. The day corresponding to this number is your lucky day of the week.

9.4.2 Block Diagram

The block diagram of the project is shown in Figure 9.22.

9.4.3 Circuit Diagram

In general, a push-button switch can be connected to microcontroller I/O pins using two methods: Active Low and Active High.

The Active Low connection is shown in Figure 9.23. When the switch is not pressed, the input pin of the microcontroller is at logic HIGH. Pressing the switch pulls down this pin to logic LOW and this change of state can be determined by the program.

The Active High connection is shown in Figure 9.24. Here, when the switch is not connected, the input of the microcontroller is at logic LOW. Pressing the switch pulls up this pin to logic HIGH.

The circuit diagram of the project is as shown in Figure 9.25. 7 LEDs are connected to PORT C pins RC0 to RC6. The push-button switch START is connected to bit 7 of PORT C (RC7) in Active Low mode. If using the EasyPIC 7 development board, Jumper J7 should be

www.it-ebooks.info

```
/*****************************************************************************************************
                                    ROTATING LEDs
                                    -------------------------

In this project 8 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller
(any other PIC microcontroller can also be used in this project).  The microcontroller is
operated from an 8MHz crystal.

In this project the LEDs rotate right to left (RB0 through RB1 and so on) and left to right
(RB7 through RB6 and so on) with one second delay between each output.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED6.C
*****************************************************************************************************/

void main()
{
  unsigned char Cnt = 1;                          // Initialise Cnt
  unsigned char Mode = 0;                         // Mode=0 right-to-left
  ANSELC = 0;                                     // Configure PORT C as digital
  TRISC = 0;                                      // Configure PORT C as outputs

  for(;;)                                         // FOREVER loop
  {
     PORTC = Cnt;                                 // Send Cnt to PORT C
     Delay_Ms(1000);                              // Delay 1 second
     if(Mode == 0)
     {
        Cnt = Cnt << 1;                           // Left shift Cnt
        if(Cnt == 0)                              // Last LED lit
        {
          Cnt = 64;
          Mode = 1;                               // Change direction
        }
     }
     else
     {
        Cnt = Cnt >> 1;                           // Right shift Cnt
        if(Cnt == 0)                              // Last LED lit
        {
          Cnt = 2;
          Mode = 0;                               // Change direction
        }
     }
  }
}
```

Figure 9.20 Listing of the modified program

www.it-ebooks.info

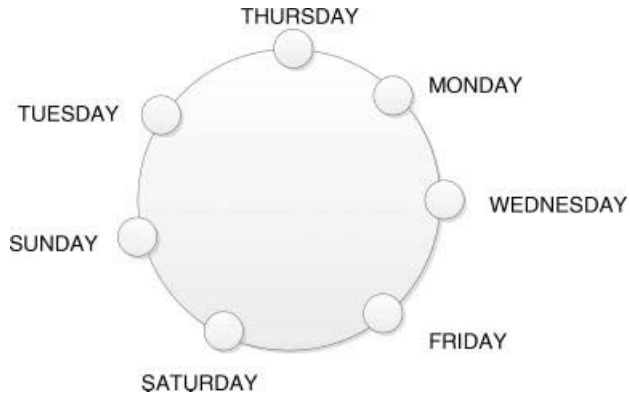


Figure 9.21 Wheel of lucky day

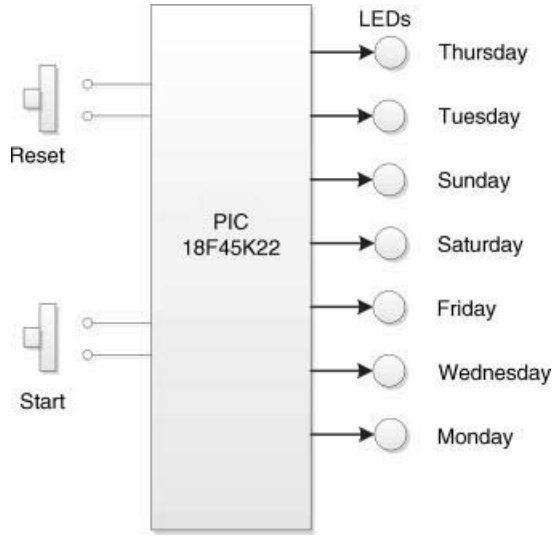


Figure 9.22 Block diagram of the project

set to the lower position and DIP switch SW7 for pin RC7 should be set to PULL-UP position. In this mode, the microcontroller input pin is normally at logic HIGH. When the button is pressed, the input pin goes to logic LOW.

9.4.4 Project PDL

The PDL of the project is shown in Figure 9.26.

9.4.5 Project Program

The program is named LED7.C and the program listing of the project is shown in Figure 9.27. At the beginning of the project, port RC7 is defined as STRT and variable *Cnt* is initialised

www.it-ebooks.info

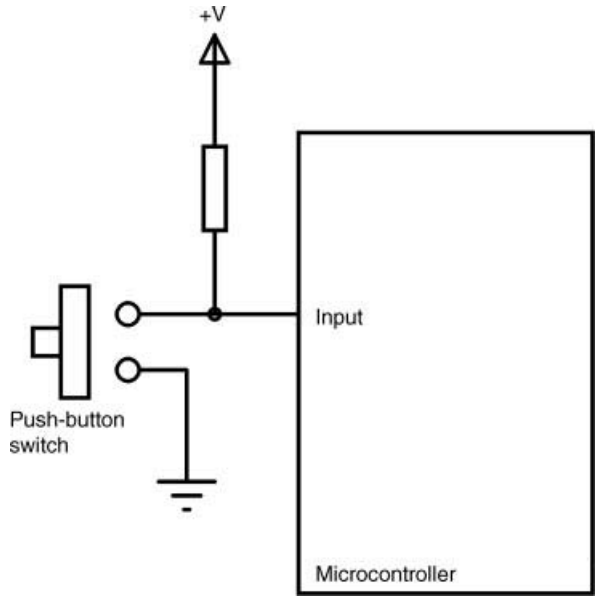


Figure 9.23 Active Low switch connection

to 1. PORT C bits RC0 to RC6 are configured as digital outputs and pin RC7 is configured as a digital input. An endless *for* loop is then constructed and the program waits until the button is pressed (until STRT is equal to 0). Once the button is pressed, an inner *for* loop is formed, where the LEDs are lit and rotated very fast, with a 50 ms delay between each output. Inside this inner *for* loop, the program checks whether or not the button is pressed (STRT is 0). When the button is pressed, the program jumps outside the inner loop and waits until the button is released. At this point, the last lit LED remains ON. After a 1 second delay, the

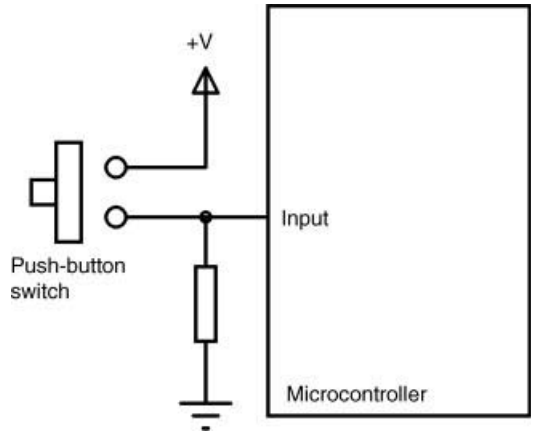


Figure 9.24 Active High switch connection

www.it-ebooks.info

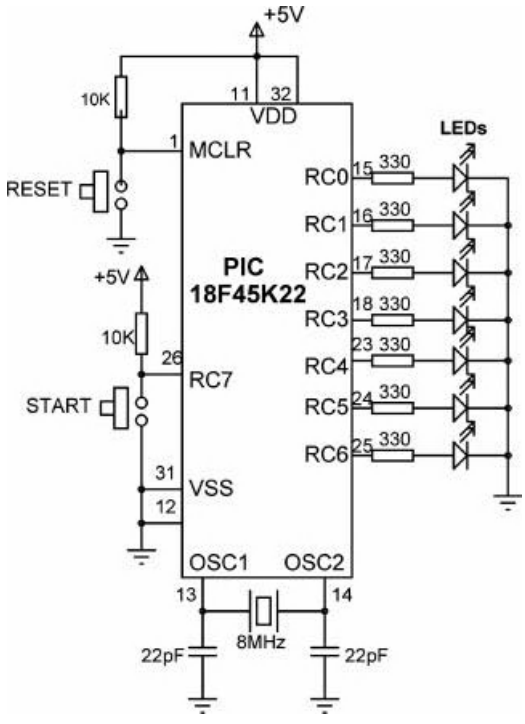


Figure 9.25 Circuit diagram of the project

```
BEGIN
    Cnt = 1
    Configure PORT C (RC0-RC6 outputs, RC7 input)
    Clear PORT C
    DO FOREVER
        Wait until button is pressed
        Wait until button is released
        DO FOREVER
            Send Cnt to PORT C
            Wait 50 milliseconds
            Shift Cnt left
            IF Cnt = 0 THEN
                Cnt = 1
            ENDIF
            IF button is pressed THEN
                EXIT inner DO loop
            ENDIF
        ENDDO
        Wait until button is released
        Wait 1 second
    ENDDO
END
```

Figure 9.26 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************************
                                WHEEL OF LUCKY DAY
                                ------------------

In this project 7 LEDs are connected to PORT C of a PIC18F45K22 typemicrocontroller
(any other PIC microcontroller can also be used in this project). In addition, a push-button
switch is connected to bit 7 of PORT C (RB7). The microcontroller is operated from an
8MHz crystal.

In this project the LEDs are constructed in the form of a circle and the days of the week
are written next toeach LED. When the push-button switch is pressed the LEDs start
rotating fast. Pressing the button stops while only one LED is lit. The day name
corresponding to this LED is your lucky day!.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED7.C
*************************************************************************************************/
#define STRT PORTC.RC7

void main()
{
  unsigned char Cnt = 1;                    // Initialise Cnt
  ANSELC = 0;                               // Configure PORT C as digital
  TRISC = 0x80;                             // RC0 - RC6 output, RC7 input
  PORTC = 0;

  for(;;)                                   // DO FOREVER
  {
    while(STRT == 1);                       // Wait until START is pressed
    while(STRT == 0);                       // Wait until button is released
    for(;;)                                 // FOREVER loop
    {
      PORTC = Cnt;                          // Send Cnt to PORT C
      Delay_Ms(50);                         // Delay 50ms
      Cnt = Cnt << 1;                       // Left shift Cnt
      if(Cnt == 0)Cnt = 1;
      if(STRT == 0)break;                   // If button is pressed
    }
    while(STRT == 0);                       // Wait until button is released
    Delay_Ms(1000);                         // Wait 1 second
  }
}
```

Figure 9.27 Program listing of the project

above process is repeated where the program waits for the button to be pressed again at the beginning of the program.

9.4.6 Switch De- Bouncing

All mechanical switches are not perfect and they exhibit some form of *contact bounce* problems. As the two metallic switch contacts are pressed together, there will be some

www.it-ebooks.info

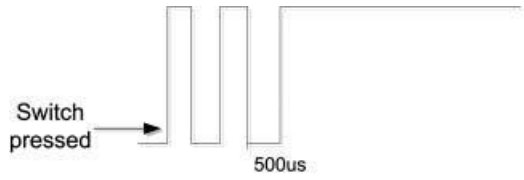


Figure 9.28 Switch contact bouncing

short time (around 10 ms) before a stable electrical contact is made. During this period the switch contacts generate many on-off contacts, as shown in Figure 9.28. This kind of behaviour is not desirable in microcontroller based circuits, as the state of the switch input may not be known exactly after a switch is pressed. The switch bouncing problems can be eliminated either in hardware or software. In hardware, usually an RC circuit is used as a filter, or a flip-flop is used to eliminate the switch contact bouncing. In software, a small delay (e.g. 10 ms) is usually inserted after the switch state changes, in order to eliminate contact bouncing.

mikroC Pro for PIC language provides a built-in function called 'button', which is used to eliminate contact bouncing problems. Use of this function is recommended when it is required to read the state of a mechanical switch. This function has the following format:

```
Button(&port, pin, time, active_state)
```

where

port and *pin* specify the port where the switch is connected to. This pin must be specified as an input pin;
time is the de-bounce period in milliseconds;
active_state is 1 or 0 and determines if the port pin is active upon logical 0 or logical 1.

A typical use of this function is given below. In this example, it is assumed that the switch is connected to bit 0 of PORT B (i.e. RB0), the de-bounce time is set to 1 ms, and the port pin is assumed to go to logic 1 when the switch is pressed:

```
Button(&PORTB, 0, 1, 1);
```

The program given in Figure 9.27 has been modified by using the *Button* function to de-bounce the switch contacts. The new program (LED8.C) listing is shown in Figure 9.29. Notice that the 1 second delay has been removed from this program, since there are no switch contact problems.

Notice that if the *Button* library is not included in your project, you may get error during the compilation time. To include this library, click View -> Library Manager from the drop-down menu of the compiler and enable the *Button* library by clicking it.

www.it-ebooks.info

```
/*************************************************************************************************
                                    WHEEL OF LUCKY DAY
                                    ------------------

In this project 7 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller
(any other PIC microcontroller can also be used in this project). In addition, a push-button
switch is connected to bit 7 of PORT C (RB7). The microcontroller is operated from an
8MHz crystal.

In this project the LEDs are constructed inthe form of a circle and the days of the week
are written next to each LED. When the push-button switch is pressed the LEDs start
rotating fast. Pressing the button stops while only one LED is lit. The day name
corresponding to this LED is your lucky day!.

In this modified program the switch contacts are de-bounced using the mikroC
"Button" function


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED8.C
*************************************************************************************************/

void main()
{
  unsigned char Cnt = 1;                          // Initialise Cnt
  ANSELC = 0;                                     // Configure PORT C as digital
  TRISC = 0x80;                                    // RC0-RC6 output, RC7 input
  PORTC = 0;

  for(;;)
  {
    while(Button(&PORTC, 7, 5, 1));               // Wait until button is pressed
    while(Button(&PORTC, 7, 5, 0));               // Wait until button is released
    for(;;)                                       // FOREVER loop
    {
        PORTC =Cnt;                               // Send Cnt to PORT C
        Delay_Ms(50);                             // Delay 100ms
        Cnt = Cnt << 1;                           // Left shift Cnt
        if(Cnt == 0)Cnt = 1;
        if(Button(&PORTC, 7, 5, 0))break;         // If button is pressed
    }
    while(Button(&PORTC, 7, 5, 0));               // Wait until button is released
  }
}
```

Figure 9.29 Modified program using the Button function

9.5 PROJECT 9.5 – Random Flashing LEDs

9.5.1 Project Description

In this project, 8 LEDs are connected to PORT C of a PIC microcontroller, as in Project 2. The LEDs turn ON and OFF randomly, with 500 ms interval between each output.

www.it-ebooks.info

9.5.2 Block Diagram

The block diagram of the project is as shown in Figure 9.12.

9.5.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.13.

9.5.4 Project PDL

The PDL of this project is very simple and is given in Figure 9.30.

9.5.5 Project Program

The program is named LED9.C and the program listing of the project is shown in Figure 9.31. In this program, a pseudorandom number generator (Gerhard's generator) function called *RandomNumber* is used to generate an integer random number between 1 and 255. This number is then sent to PORT C of the microcontroller every 500 ms. The LEDs flash randomly, giving nice patterns of display. The random number generator function requires a seed and the maximum number to be generated as its arguments.

9.6 PROJECT 9.6 – LED Dice

9.6.1 Project Description

In this project, 7 LEDs are organised in the form of a dice. A push-button switch (START) is used, such that when the switch is pressed, a random number is displayed between 1 and 6 by

```
BEGIN
        Configure PORT C
        DO FOREVER
                Call RandomNumber
                Send the random number to PORT C
                Wait 0.5 second
        ENDDO
END

BEGIN/RandomNumber
        Generate a random number between 1 and 255
END/RandomNumber
```

Figure 9.30 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************************
                                    RANDOM FLASHIMG LEDs
                                    --------------------

In this project 8 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller.
(any other PIC microcontroller can also be used in this project) The microcontroller is
operated from an 8MHz crystal.

A pseudorandom number generator (Gerhard's generator) function is used to generate
an integer number between 1 and 255. This number is then sent to PORT C to flash
the LEDs ON and OFF randomly.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED9.C
*************************************************************************************************/

//
// This function generates a pseudorandom integer number between 1 and Lim
// A seed is given to the generator to start with
//
unsigned char RandomNumber(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
 }


//
// Start of main program
//
void main()
{
   unsigned char J, seed = 1;                    // Initialise the seed
   ANSELC = 0;                                   // Configure PORT C as digital
   TRISC = 0;                                    // PORT C is output

   for(;;)                                       // DO FOREVER
   {
      J = RandomNumber(255, seed);               // Generate a number between 1 and 255
      PORTC = J;                                 // Send number to PORT C
      Delay_Ms(500);                             // Wait 0.5 second
   }
}
```

Figure 9.31 Program listing of the project

www.it-ebooks.info

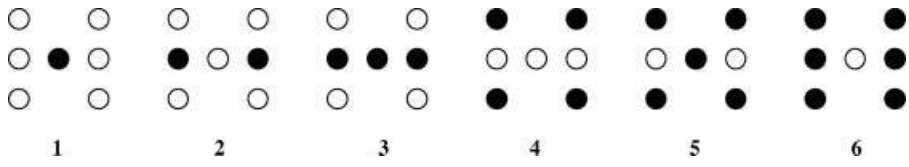


Figure 9.32 Dice numbers

the LEDs to imitate a real dice. The LEDs are turned OFF after 5 seconds to indicate that the system is ready and the user can press the button for a new dice number.

Figure 9.32 shows the LED organisation and the corresponding dice numbers.

9.6.2 Block Diagram

The block diagram of the project is shown in Figure 9.33.

9.6.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 9.34. The START switch is connected to bit 7 of PORT C. The 7 LED s are connected to bits RC0 to RC6 of PORT C as follows:

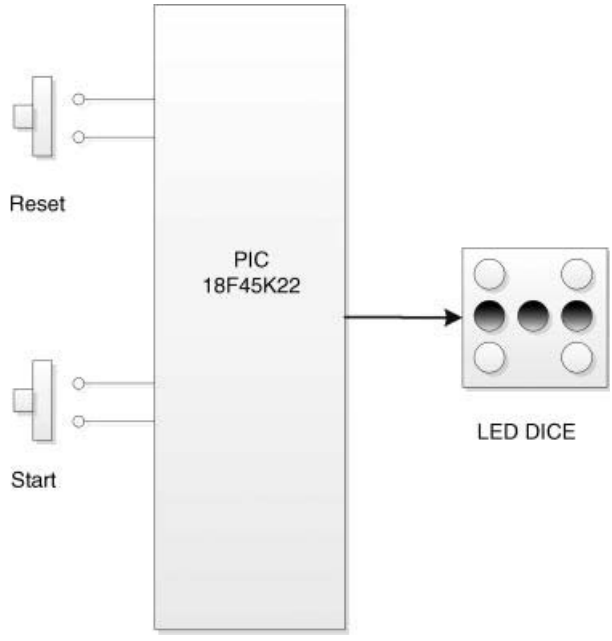


Figure 9.33 Block diagram of the project.

www.it-ebooks.info

LED	PORT C pin
L1	RC0
L2	RC1
L3	RC2
L4	RC3
L5	RC4
L6	RC5
L7	RC6

The microcontroller is operated from an 8 MHz crystal. Table 9.1 shows the relationship between a dice number and the corresponding LEDs to be turned ON to show the numbers. For example, to display number 1, we have to turn ON only the middle LED, that is L4. Similarly, to display number 3, we have to turn on LEDs L2, L4 and L6. The last column of Table 9.1 shows the hexadecimal number to be sent to PORT C to display a given dice number. For example, to display number 4, we have to send hexadecimal number 0×55 to PORT C. If using the EasyPIC 7 development board, Jumper J7 should be set to the lower position and DIP switch SW7 for pin RC7 should be set to PULL-UP position. In this mode, the microcontroller input pin is normally at logic HIGH. When the button is pressed, the input pin goes to logic LOW.

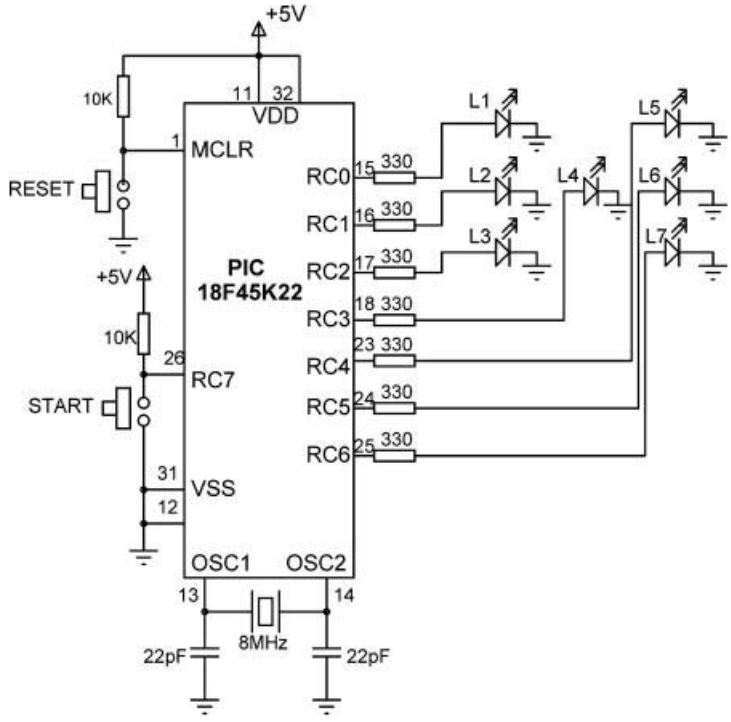


Figure 9.34 Circuit diagram of the project

www.it-ebooks.info

Table 9.1 Data to be sent to PORT C for a given dice number

Required Number	LEDs to be Turned On	PORT C Data (Hex)
1	L4	0×08
2	L2, L6	0×22
3	L2, L4, L6	$0 \times 2A$
4	L1, L3, L5, L7	0×55
5	L1, L3, L4 L5, L7	$0 \times 5D$
6	L1, L2, L3, L5, L6, L7	0×77

9.6.4 Project PDL

The PDL of this project is given in Figure 9.35. As you can see from the PDL, the dice number is created using a pseudorandom number generator, as in the previous project.

9.6.5 Project Program

The project program listing (LED10.C) is shown in Figure 9.36. PORT C bit 7 (RC7) is configured as a digital input port and bits RC0 to RC6 are configured as output ports. An array called DICE is created and initialised with the dice patterns corresponding to dice numbers. Notice that DICE[0] is not used and is set to 0. When the START button is pressed, function *RandomNumber* is called to generate an integer pseudorandom number (in variable *J*) between 1 and 6. The bit pattern corresponding to this number is read from array DICE (in variable *No*) and sent to PORT C as a hexadecimal number in order to display the

```
BEGIN
        Create DICE table
        Configure PORT C
        Clear PORT C
        DO FOREVER
                IF button is pressed THEN
                        CALL RandomNumber
                        Get LED pattern for this number from DICE table
                        Send pattern to PORT C
                        Wait 5 seconds
                        Clear PORT C
                ENDIF
        ENDDO
END


BEGIN/RandomNumber
        Get a random number between 1 and 6
END/RandomNumber
```

Figure 9.35 PDL of the project

www.it-ebooks.info

```
/*****************************************************************************************************
                                    LED DICE
                                    --------

In this project 7 LEDs are connected to PORT C of a PIC18F45K22 type microcontroller.
(any other PIC microcontroller can also be used in this project) in the form of a dice.
In addition, a push-button switch is connected to bit 7 of PORT C (RC7). The
microcontroller is operated from an 8MHz crystal.

When the button (called START) is pressed, a pseudorandom number is generated
between 1 and 6 and the LEDs are lit to show this number by imitating a dice.

Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED10.C
*****************************************************************************************************/
#define START PORTC.RC7

//
// This function generates a pseudorandon integer number between
// 1 and Lim. A seed is given to the generator to start with
//
unsigned char RandomNumber(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
 }

//
// Start of main program
//
void main()
{
  unsigned char J, No, seed = 1;                    // Initialise the seed
  unsigned char DICE[] = {0,0X08,0X22,0X2A,0X55,0X5D,0x77};
  ANSELC = 0;                                       // Configure PORT C as digital
  TRISC = 0x80;                                     // RC7 is input, RC0-RC6 are outputs

  PORTC = 0;                                        // Clear PORT C
  for(;;)                                           // DO FOREVER
  {
    if(START == 0)                                  // If START button is pressed
    {
      J = RandomNumber(6, seed);                    // Generate number between 1–6
      No = DICE[J];                                 // Hex number corresponding to
                                                    //  this number
      PORTC = No;                                   // Send to PORT C
      Delay_Ms(5000);                               // Wait 5 seconds
      PORTC = 0;                                    // Clear PORT C for the next time
    }
  }
```

Figure 9.36 Program listing of the project

www.it-ebooks.info

imitated dice number on the LEDs. The dice number is displayed for 5 seconds and then the LEDs turn OFF to indicate that the system is ready to generate a new number.

9.6.6 Suggestions for Further Development

Some of the games (e.g. backgammon) are played with two dices. The design given in this project can be modified by adding another set of 7 LEDs for the second dice. For example, the first dice can be driven from PORT C, the second one from PORT D.

9.7 PROJECT 9.7 – Connecting more than one LED to a Port Pin

9.7.1 Project Description

There are some applications where we may want to connect many LEDs to a microcontroller, but where only one LED is required to be ON at any time. One such application is the design of a Roulette board, which requires 37 LEDs, and only one LED is ON at any time. In principle, we can use a high-end microcontroller with many I/O pins for such applications. But, as we shall see in this project, for low-cost applications it is preferable to use a cheaper low-end microcontroller and share the I/O pins. This approach requires less wiring and less number of current limiting resistors, resulting in a much lower cost.

As shown in Figure 9.37, connecting two LEDs to an I/O port is very easy. One LED is connected in current sourcing mode, while the other LED is connected in current sinking mode. When the port pin is at logic LOW, LEDA is turned ON, when it is at logic HIGH, LEDB is turned ON.

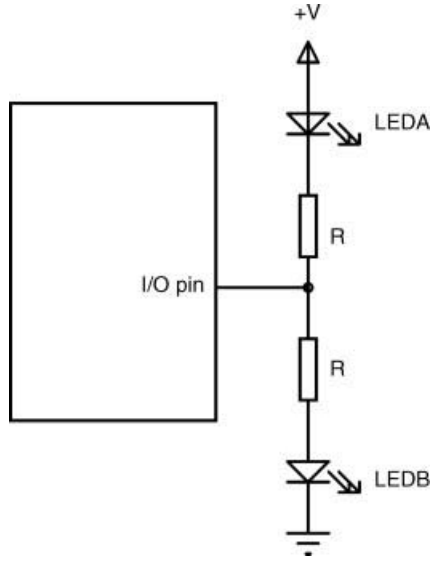


Figure 9.37 Connecting 2 LEDs to a port pin

www.it-ebooks.info

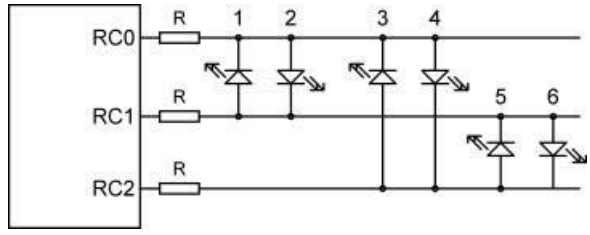


Figure 9.38 Connecting 6 LEDs to 3 I/O pins

It is possible to connect more than two LEDs to a port pin, where only one LED is ON at any time. Here, we connect two LEDs between each pair of I/O pins, where the LEDs are connected in opposite directions. Thus, with 3 I/O pins, we can have up to 6 LEDs, with 4 I/O pins we can have up to 12 LEDs, and so on. The maximum number of LEDs that can be connected can be calculated from:

$$N * (N - 1) \tag{9.1}$$

where N is the number of I/O pins used. For example, with a complete port (8 I/O pins) we can connect up to 56 LEDs to the microcontroller. The number of current limiting resistors is equal to the number of I/O pins used. Thus, 56 LEDs can be driven from a port with only 8 current limiting resistors.

Figure 9.38 shows the circuit where 3 I/O pins are used and up to 6 LEDs are connected to the microcontroller. Notice that two LEDs are connected between each pair of I/O lines, where the LEDs are connected in opposite polarities. In general, this circuit can be expanded to any number of I/O pins and any number of LEDs.

The operation of the circuit in Figure 9.38 is explained below:

As an example, let us assume that line RC1 is disconnected. Now, to turn ON LED 4, RC0 has to be logic 1 and RC2 has to be logic 0, so that the LED is forward biased and current flows from RC0 to RC2. You will also notice that LEDs 2 and 6 will also be forward biased. But because these LEDs are in series, they will require a minimum of 4 V to operate (assuming red LEDs): and considering the voltage drop across the current limiting resistors also, there is not enough current to operate these LEDs and they will both be OFF. But what we have just described will only work if line RC1 is disconnected. If RC1 = 0, then LED 2 will also be ON. On the other hand, if RC1 = 1, then LED 6 will also be ON. The question is now, how can we disconnect line RC1? The easiest way is to make line RC1 an input port. When a port is in input mode, it is in high-impedance state so that no current flows in or out of the port, effectively disconnecting the port pin from the circuit. It is important to notice that when we use this technique and we have different colour LEDs, then the forward voltage drops of each LED should be similar (e.g. we can easily mix red and green LEDs). The reason for this can be explained by looking at Figure 9.38. If we assume that LED 4 has a larger than normal forward voltage (e.g blue or white LED), then the voltage dropped across LEDs 2 and 6 may just be high enough to turn them ON.

Now, based on this technique, we can turn ON each LED individually. Table 9.2 shows what the state of each port pin should be to turn ON the LEDs in Figure 9.38. Notice here

www.it-ebooks.info

Table 9.2 LED control table for Figure 9.38

1	2	3	4	5	6	RC2	RC1	RC0
OFF	OFF	OFF	OFF	OFF	OFF	0	0	0
ON	OFF	OFF	OFF	OFF	OFF	Z	1	0
OFF	**ON**	OFF	OFF	OFF	OFF	Z	0	1
OFF	OFF	**ON**	OFF	OFF	OFF	1	Z	0
OFF	OFF	OFF	**ON**	OFF	OFF	0	Z	1
OFF	OFF	OFF	OFF	**ON**	OFF	1	0	Z
OFF	OFF	OFF	OFF	OFF	**ON**	0	1	Z
OFF	OFF	OFF	OFF	OFF	OFF	1	1	1

that Z indicates that the port is in input mode. As an example, to turn ON LED 5, the required port settings are: RC0 = input mode, RC1 = 0, RC2 = 1.

We shall now create a project to control 6 LEDs connected to only 3 ports of a PIC microcontroller. In this project, the LEDs will rotate flashing with 1 second delay between each output.

9.7.2 Block Diagram

The block diagram of the project is shown in Figure 9.39.

9.7.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 9.40. The 6 LEDs are connected to port pins RC0, RC1 and RC2. Notice that when an LED is ON, the current flows through two current limiting resistors. As a result, smaller resistors are required (e.g. 150 ohm or smaller).

The LEDs are turned ON in the following sequence: 1, 2, 3, 4 5, 6, 1, 2, . . .

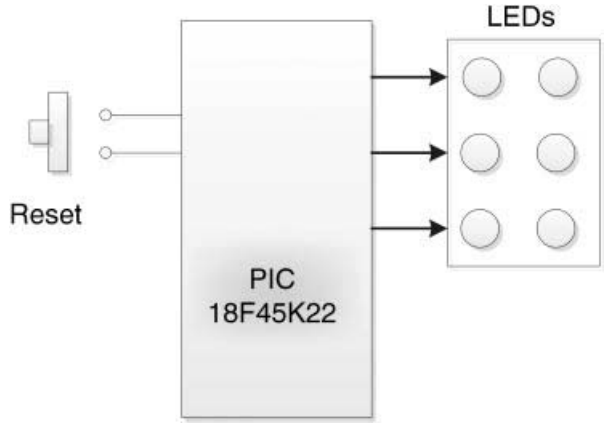


Figure 9.39 Block diagram of the project

www.it-ebooks.info

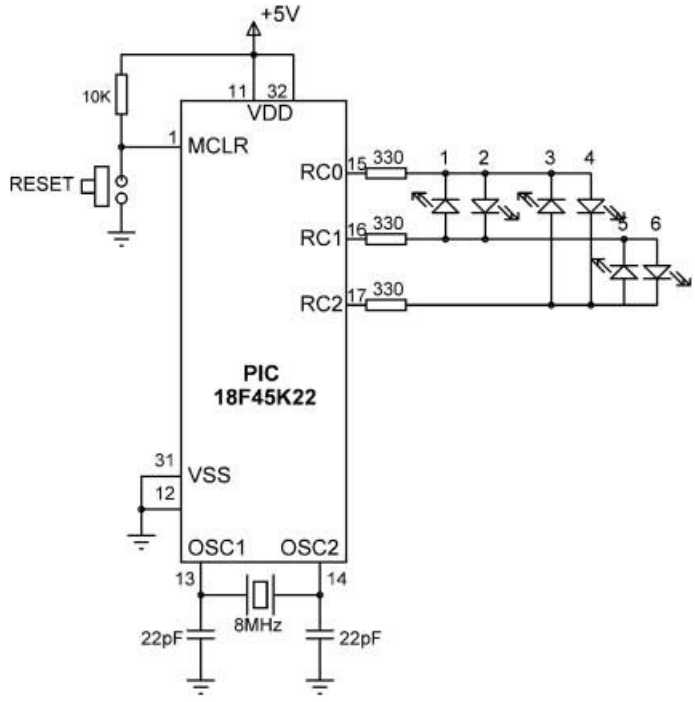


Figure 9.40 Circuit diagram of the project

9.7.4 Project PDL

The PDL of this project is given in Figure 9.41.

9.7.5 Project Program

The project program listing (LED11.C) is shown in Figure 9.42. All port pins are initially configured as outputs and the LEDs are turned OFF. Function *TURNON* implements Table 9.2 using a switch statement and turns ON a given LED. Z is defined as 1 and is used to set a port pin into input mode. Variable *j* stores the LED number turned ON and takes values between 1 and 6. This function is called by passing the number of the LED (1 to 6) to be turned ON at any time. Passing a 0 clears, that is turns OFF all the LEDs.

9.7.6 Suggestions for Further Development

The project given in this section can be developed further, for example by designing an LED based Roulette. This project will require 37 LEDs to be connected to the microcontroller and, using the technique described in this section, 7 I/O pins allow up to 42 LEDs to be connected easily with 7 current limiting resistors.

www.it-ebooks.info

```
BEGIN
        Set LED number to 1
        Configure PORT C
        Clear PORT C to start with
        DO FOREVER
                CALL TURNON and pass LED number as an argument
                Wait 1 second
                Increment LED number (1 to 6)
        ENDDO
END


BEGIN/TURNON
        Turn ON the LED whose number is received as an argument
END/TURNON
```

Figure 9.41 PDL of the project

9.8 PROJECT 9.8 – Changing the Brightness of LEDs

9.8.1 Project Description

In some applications we may want to change the brightness of an LED. Perhaps the easiest way to do this is to vary the current through the LED by using a potentiometer. By keeping the supply voltage the same, we can connect a potentiometer in series with the LED and by varying the potentiometer we effectively change the current through the LED, which in turn changes its brightness. But in microcontroller applications, we wish to write a program and then change the LED brightness by running the program. Before we can do this, it is worthwhile looking at the theory briefly on how we can change the power delivered to an LED.

One of the common techniques used to change the power delivered to an LED is to use a Pulse-Width-Modulated (PWM) signal. This is basically a positive square wave signal where the ON to OFF period can be changed by software. Let us initially consider applying a square wave signal to the LED with equal ON and OFF periods. If the duration of these signals are long (e.g. 1 s), then we will see the LED flashing ON and OFF. If we now decrease the duration to say around 100 ms, we will see rapid flashing. If we continue to decrease the duration to around 20 ms, the LED will seem to have stopped flashing and we will see reduced brightness. This is because the LED brightness is now determined by the average current produced by the square wave signal. Now, if we change the ON to OFF ratio of the signal, we will see the brightness of the LED changing. This is the principle of controlling the power delivered to the LED using the PWM signal.

Figure 9.43 shows a typical PWM signal. The ratio of the ON period (also called MARK) to the OFF period (also called SPACE) is known as the Duty-Cycle of the PWM signal. The duty-cycle is quoted as a percentage and is calculated as:

$$D = (T_{on}/T) \times 100\% \tag{9.2}$$

where D is the duty-cycle, T is the period and T_{on} is the ON time.

www.it-ebooks.info

```
/*************************************************************************************************
                MORE THAN 1 LED ON A PORT PIN
                -----------------------------------------------------

In this project 6 LEDs are connected to pins RC0, RC1 and RC2 of PORT C. Using only
3 pins the 6 LEDs are turned ON and OFF in a rotating fashion (see text for more
information).

The program shows how large number of LEDs can easily be connected to I/O ports.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED11.C
*************************************************************************************************/
#define Z 1

//
// Define LED connections to PORTT C
//
sbit RC0bit at RC0_bit;
sbit RC1bit at RC1_bit;
sbit RC2bit at RC2_bit;
//
// Define direction registers of used PORT C bits
//
sbit RC0_Direction at TRISC0_bit;
sbit RC1_Direction at TRISC1_bit;
sbit RC2_Direction at TRISC2_bit;


//
// This function sends the correct signals to turn ON the required LED. Only 1 LED
// is turned ON at any time. "0" turns OFF all LEDs. When theport direction bit is
// set to Z (= 1) then the port pin is an input pin. In this example 6 LEDs are connected
// to the microcontroller PORT C pins
//
void TURNON(unsigned char No)
{
  TRISC = 0;
  switch(No)
  {
   case 0:
        RC0bit = 0; RC1bit = 0; RC2bit = 0;
        break;
   case 1:
        RC0bit = 0; RC1bit = 1; RC2_Direction = Z;
        break;
   case 2:
        RC0bit = 1; RC1bit = 0; RC2_Direction = Z;
        break;
```

Figure 9.42 Program listing of the project

www.it-ebooks.info

```
    case 3:
        RC0bit = 0; RC1_Direction = Z; RC2bit = 1;
        break;
    case 4:
        RC0bit = 1; RC1_Direction = Z; RC2bit = 0;
        break;
    case 5:
        RC0_Direction = Z; RC1bit = 0; RC2bit = 1;
        break;
    case 6:
        RC0_Direction = Z; RC1bit = 1; RC2bit = 0;
  }
}


//
// Start of main program
//
void main()
{
  unsigned char j = 1;                    // Initialise j
  ANSELC = 0;                             // Configure PORT C as digital
  TRISC = 0;                              // PORT C as output to start with

  PORTC = 0;                              // Clear PORT C
  for(;;)                                 // DO FOREVER
  {
    TURNON(j);                            // Turn ON LED j
    Delay_Ms(1000);                       // Wait 1 second
    j++;                                  // Increment j
    if(j == 7)j = 1;                      // Back to 1st LED
  }
}
```

Figure 9.42 (*Continued*)

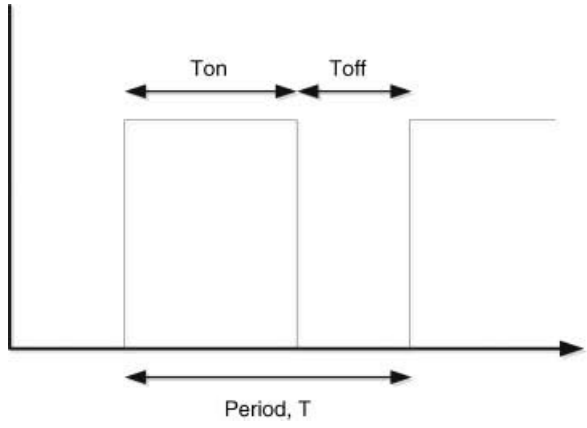


Figure 9.43 PWM signal

www.it-ebooks.info

When the period is around 20 ms, the human eye will not see the flashes and a continuous brightness will be seen. In this section we will assume a 20 ms (20 000 μs) period. The required ON and OFF times for a given duty-cycle can be calculated as follows:

$$T_{on} = D \times T/100 \tag{9.3}$$

or

$$T_{on} = 200D \tag{9.4}$$

and

$$T_{off} = T - T_{on} \tag{9.5}$$

or

$$T_{off} = 200\,(100 - D) \tag{9.6}$$

where T_{on} and T_{off} are in μs. Equations 9.4 and 9.6 can be used to calculate the required ON and OFF times. For example, for a 75% duty-cycle, the required ON and OFF times are

$$\begin{aligned} T_{on} &= 200D = 200 \times 75 = 15,000\,\text{ms} \\ T_{off} &= 200\,(100 - D) = 200 \times 25 = 5,000\,\text{ms} \end{aligned} \tag{9.7}$$

As the duty-cycle is varied, the power delivered to the LED and hence the brightness of the LED changes. We can divide the period into equal steps for simplicity and the size of a step is known as the resolution of the PWM signal. Here, the period is 20 000 μs and it is reasonable to divide this into 20 steps, each step being 1000 μs. That is the resolution of our signal will be 1000 μs. This corresponds to 20 steps. That is, we should be able to change the steps and see the changes in the brightness of our LED. Since we have 20 steps, each step corresponds to a change of 5%.

There are several techniques that we can use to generate a PWM signal using a microcontroller. Some of the commonly used techniques are

- using delays in a program;
- using timer interrupts;
- using the built-in PWM module of the microcontroller.

Each technique will be described as a project in the following sections.

9.8.1.1 Using Delays to Generate a PWM Signal

Perhaps the easiest technique to generate a PWM signal is by using delays in a program. Here, we can set up an endless loop and inside this loop we can send a logic HIGH from a microcontroller port, then wait for T_{on} microseconds. Then, we can send a logic LOW signal

www.it-ebooks.info

from the same port and wait for T_{off} microseconds. This process can be repeated continuously to generate a continuous PWM signal. The disadvantage of this technique is that the microcontroller is dedicated to generating a PWM signal and it cannot do any other useful work. In addition, the timing of the signal is not accurate.

In this project, the brightness of an LED is controlled by increasing the ON time every second, in steps of 1 ms, from 1 to 20 ms.

9.8.2 Block Diagram

The block diagram of the project is as shown in Figure 9.3.

9.8.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.44, where only a single LED is connected to port pin RC0 of the microcontroller.

9.8.4 Project PDL

The PDL of this project is given in Figure 9.44.

9.8.5 Project Program

The program is named LED12.C and the program listing of the project is shown in Figure 9.45. The time units used in the program are milliseconds rather than microseconds, as the Delay_us function only accepts constant numbers, not variables. In this program, Vdelay_Ms function is used to create the require delays in milliseconds.

```
BEGIN
    Configure PORT C
    Set On time to 0ms
    DO FOREVER
        Calculate ON time
        Calculate OFF time
        Send 1 to RC0
        Wait for ON time
        Send 0 to RC0
        Wait for OFF time
        IF 1 second elapsed THEN
            Increase ON time by 1ms
            IF ON time > 20ms THEN
                Set ON time to 0ms
            ENDIF
        ENDIF
    ENDDO
END
```

Figure 9.44 PDL of the project

www.it-ebooks.info

```
/*****************************************************************************************************
                        CHANGE LED BRIGHTNESS
                        ---------------------------------------

In this project an LED is connected to pin RC0 of aPIC microcontroller. The brightness
of the LED is increased every second in 20 steps (5% each second) by sending a PWM
waveform to the LED. i.e. the ON time is increased every second from 0ms to 20ms,
in steps of 1ms.


Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED12.C
*****************************************************************************************************/
#define PWM PORTC.RC0

void main()
{
  unsigned char j = 0;
  unsigned char b = 0;
  unsigned int ONtime, OFFtime;                 // Waveform ON and OFF times

  ANSELC = 0;                                   // Configure PORT C as digital
  TRISC = 0;                                    // PORT C as output to start with

  for(;;)                                       // DO FOREVER
  {
    ONtime = b;                                 // PWM ON time (in ms)
    OFFtime = 20 - ONtime;                      // PWM OFF time (in ms)
    PWM = 1;                                    // Set to 1
    Vdelay_Ms(ONtime);                          // Delay for ON time
    PWM = 0;                                    // Set to 0
    Vdelay_Ms(OFFtime);                         // Delay for OFF time
    j++;
    if(j == 50)                                 // If one second elapsed, change brigthness
    {
     j = 0;
     b++;                                       // Increase ON time by 1ms
     if(b > 20)b = 0;
    }
  }
}
```

Figure 9.45 Program listing of the project

At the beginning of the program, the ON time variable *ONtime* is initialised to 0. Then, an endless loop is formed using the *for* statement. Inside this loop, the ON and the OFF times are calculated in milliseconds. Logic 1 is sent to port pin RC0 and the program waits for the duration of the ON time. Then, logic 0 is sent and the program waits for the duration of the OFF time. The ON time is incremented by 1 ms approximately every second. After 50 iterations, the elapsed time is assumed to be 1 second ($20\,\text{ms} \times 50 = 1000\,\text{ms}$, i.e. 1 s). When the ON time reaches 20, it is reset back to 0. When the program is run, the brightness of the LED should increase every second. After about 20 seconds, the LED will be OFF and the above process will be repeated.

www.it-ebooks.info

9.8.6 Using Timer Interrupts to Generate a PWM Signal

The advantage of using timer interrupts to generate a PWM signal is that the signal is generated in the background and the processor is free to do other tasks. Here, we will set a high-priority timer interrupt using the TMR0 module. The processor will jump to the interrupt service routine (ISR) every 1000 μs. The PWM ON or OFF times will be stored in variables and will be decremented inside the ISR every time an interrupt occurs. When one of the variables reach zero, the output will be toggled. That is, 0 will change to 1, and 1 will change to 0, and this variable will be re-loaded to its initial value. In this project, the PWM ON and OFF times will be set to 15 000 and 5000 μs, respectively.

9.8.6.1 Block Diagram

The block diagram of the project is as shown in Figure 9.3.

9.8.6.2 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.4, where only a single LED is connected to port pin RC0 of the microcontroller.

9.8.6.3 Project PDL

The PDL of this project is given in Figure 9.46.

```
BEGIN
        Configure PORT C as digital and output
        Configure TMR0 to generate interrupts every 1,000μs
        Define PWM ON and OFF times
        Wait for interrupts
END


BEGIN/INTERRUPT
        Reload TMR0 register
        IF output is 1 THEN
                Decrement ON time counter
                IF end of ON time THEN
                        Set output to 0
                        Reload ON time counter
                ENDIF
        ELSE
                Decrement OFF time counter
                IF end of OFF time THEN
                        Set output to 1
                        Reload OFF time counter
                ENDIF
        ENDIF
        Enable TMR0 interrupt flag
END/INTERRUPT
```

Figure 9.46 PDL of the project

www.it-ebooks.info

9.8.6.4 Project Program

The program is named LED13.C and the program listing of the project is given in Figure 9.47. At the beginning of the program, the period of the PWM wave is defined as 20 000 μs, and PORT C is configured as a digital output port.

The interrupt priority feature is enabled by setting IPEN = 1 and the timer TMR0 interrupts are set to high-priority by setting TMR0IP = 1. The TMR0 interrupt flag is cleared by setting TMR0IF = 0, the timer is set to operate in 8-bit mode and with a prescaler value of 64. As described in Chapter 2, the required timer value to generate interrupts every 1000 μs can be calculated from:

$$\text{TMR0L} = 256 - 1000/(4 * \text{Tosc} * \text{Prescaler}) \tag{9.8}$$

or

$$\text{TMP0L} = 256 - 1000/(0.5 * 64) \tag{9.9}$$

giving TMR0L = 224.75, the nearest integer is selected, that is TMR0L = 225.

TMR0 interrupts are enabled by setting TMR0IE = 1. Finally, global interrupts are enabled by setting GIEH = 1. The timer is now ready to generate interrupts every 1000 μs. In the main program, the PWM On and OFF times are set to 15 000 and 5000 μs, respectively. The main program then enters an endless loop and waits for timer interrupts to occur.

When an interrupt occurs, the program jumps to the ISR declared as function *interrupt*. Here, the first task is to re-load the timer register TMR0L. If the output is 1, the ON time is decremented by 1000 μs. When the ON time reaches 0, the output is changed to 0 and the counter is reloaded. If, on the other hand, the output is 0, the OFF time is decremented by 1000 μs. When the OFF time reaches 0, the output is changed to 1 and the counter is reloaded. Just before exiting the ISR, further timer interrupts are re-enabled by clearing the timer interrupt flag TMR0IF.

The PWM output waveform generated by the program in Figure 9.47 is shown in Figure 9.48.

9.8.7 Changing the Brightness Continuously with PWM

The program given in Figure 9.47 can be modified such that the brightness of the LED can be increased continuously, for example every second. When the LED is fully bright, it can be turned OFF and the above process can be repeated. The new program listing (LED14.C) is shown in Figure 9.49. Here, the main program has been changed by adding 2 seconds of delay and then increasing the brightness every second. In Figure 9.49, only the main program is shown, as the ISR is as before.

9.8.8 Suggestion for Further Developments

In some applications we may want to control the brightness of multiple LEDs. If all LEDs are to have the same brightness, then we could simply make the following changes to the program:

www.it-ebooks.info

```
/*************************************************************************************************
                CHANGE LED BRIGHTNESS USING PWM
                ---------------------------------------------------------

In this project an LED is connected to pin RC0 of a PIC microcontroller. A PWM wave is
generated with an ON time of 15,000us and OFF time of 5,000 us. The period of the
waveform is 20,000us.

Timer TMR0 interrupts are used to generate the PWM wave. The timer interrupts every
1,000us and two counters are used to determine what the output should be: PWMOn
and PWMoff. These counters store the ON and OFF times in microseconds. They are
decremented by 1,000 every time a timer interrupt occurs. When a counter reached zero,
the output is toggled and the counter is re-loaded.

Timer TMR0 is operated in the following mode:

Mode: 8-bit
Interrupt: High-priority
Prescaler: 64
TMR0 Count: 225

Author:  Dogan Ibrahim
Date:    October, 2011
File:    LED13.C
*************************************************************************************************/
#define PWM PORTC.RC0

unsigned int ONtime, OFFtime, PWMon, PWMoff;

//
// This is the Interrupt Service Routine (ISR). The program jumps to
// this routine every 1,000us.
//
void interrupt(void)
{
  TMR0L = 225;                                  // Re-load TMR0L

  if(PWM == 1)                                  // If output is 1
  {
    PWMon = PWMon-1000;                         // Decrement ON time
    if(PWMon <= 0)                              // IF end of ON time
    {
      PWM = 0;                                  // Set output to 0
      PWMon = ONtime;                           // Reload counter
    }
  }
  else
  {                                             // If output is 0
   PWMoff = PWMoff-1000;                        // Decrement OFF time
   if(PWMoff <= 0)                              // IF end of OFF time
    {
```

Figure 9.47 Program listing of the project

www.it-ebooks.info

```
      PWM = 1;                                          // Set output to 1
      PWMoff = OFFtime;                                 // Reload OFF time
    }
  }

  TMR0IF_bit = 0;                                       // Clear TMR0 interrupt flag
}


//
// Main program
//
void main()
{
  unsigned intPeriod = 20000;                           // Period in microseconds

  ANSELC = 0;                                           // Configure PORT C as digital
  TRISC = 0;                                            // PORT C as output
//
// Configure TMR0 to generate interrupts at every 1000us
//
  IPEN_bit = 1;                                         // Enable priority based interrupts
  TMR0IP_bit = 1;                                       // TMR0 in high priority
  TMR0IF_bit = 0;                                       // Clear TMR0 interrupt flag
  TMR0L = 225;                                          // Load TMR0L
  T0CON = 0b11000101;                                   // TMR0 in 8 bit mode, prescaler=64
  TMR0IE_bit = 1;                                       // Enable TMR0 interrupts
  PWM = 0;                                              // Output 0 to start with
  GIEH_bit = 1;                                         // Enable high priority interrupts

  ONtime = 15000;
  OFFtime = Period - ONtime;
  PWMon = ONtime;
  PWMoff = OFFtime;

  for(;;)                                               // Wait for interrupts
  {

  }

}
```

Figure 9.47 (*Continued*)

```
Change #define PWM PORTC.RC0  to #define PWM PORTC
Change if(PWM == 1)           to if(PWM == 0xFF)
Change PWM = 1                to PWM = 0xFF
```

If, on the other hand, we want to control the brightness of each LED independently, then we could extend the ISR by creating independent counters for each LED and then sending a 1 or 0 to each LED whenever its counter becomes 0.

www.it-ebooks.info

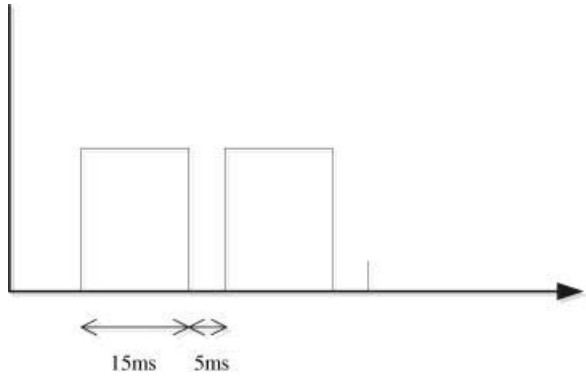


Figure 9.48 Generated PWM waveform

```
void main()
{
  unsigned int Period = 20000;                    // Period in microseconds

  ANSELC = 0;                                     // Configure PORT C as digital
  TRISC = 0;                                      // PORT C as output
//
// Configure TMR0 to generate interrupts at every 1000us
//
  IPEN_bit = 1;                                   // Enable priority based interrupts
  TMR0IP_bit = 1;                                 // TMR0 in high priority
  TMR0IF_bit = 0;                                 // Clear TMR0 interrupt flag
  TMR0L = 225;                                    // Load TMR0L
  T0CON = 0b11000101;                             // TMR0 in 8 bit mode, presasler=64
  TMR0IE_bit = 1;                                 // Enable TMR0 interrupts
  PWM = 0;
  GIEH_bit = 1;                                   // Enable high priority interrupts


  ONtime = 1000;
  OFFtime = Period - ONtime;
  PWMon = ONtime;
  PWMoff = OFFtime;

  for(;;)
  {
   Delay_Ms(2000);
   Ontime = ONtime + 1000;                        // Increase brightness
   if(ONtime == 20000) ONtime = 1000;
  }

}
```

Figure 9.49 Modified program to change the brightness

www.it-ebooks.info

9.8.9 *Using the Built-in Microcontroller PWM Module*

The PIC18 microcontroller family has built-in PWM modules (see Chapter 2) that can be used to generate PWM waveforms. PWM module makes use of ports CCP1, CCP2, CCP3, and so on, of the microcontroller. But unfortunately this module cannot be used to generate PWM signals with large periods such as 20 ms. In this project we shall be using the CCP1 pin and connect an LED to this pin via a current limiting resistor and we will generate a PWM signal with a period of 2000 μs, ON time (duty cycle) of 500 μs, and OFF time of 1500 μs. The advantage of using the PWM module is that, like the interrupts, the waveform is generated in the background and the processor is free to do other tasks while the PWM signal is being generated.

9.8.9.1 Block Diagram

The block diagram of the project is as shown in Figure 9.3.

9.8.9.2 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.4, except that the LED is connected to port pin CCP1 (pin RC2 in PIC18F45K22 microcontroller) via a current limiting resistor.

9.8.9.3 Project PDL

The PDL of this project is given in Figure 9.50.

9.8.9.4 Project Program

The program is named LED15.C and the program listing of the project is shown in Figure 9.51.

As described in Chapter 2, the value to be loaded into Timer 2 register can be calculated as

$$PR2 = \frac{PWM\ period}{TMR2PS * 4 * Tosc} - 1 \tag{9.10}$$

where

PR2 is the value to be loaded into Timer 2 register;
TMR2PS is the Timer 2 prescaler value;
Tosc is the clock oscillator period (in seconds).

```
BEGIN
    Configure PORT C digital and output
    Configure Timer 2 for PWM module
    Load Timer register PR2 with period
    Load duty cycle into CCPR1L and CCP1CON
    Wait here forever
END
```

Figure 9.50 PDL of the project

www.it-ebooks.info

```
/*********************************************************************************************
                CHANGE LED BRIGHTNESS USING THE PWM MODULE
                ------------------------------------------

In this project an LED is connected to pin CCP1 of a PIC18F45K22  microcontroller.
A PWM wave is generated with an ON time of 500us and OFF time of 1,5000us.
The period of the waveform is 2,000us. The microcontroller is operated from an
8MHz crystal


Author:  Dogan Ibrahim
Date:    November, 2011
File:    LED15.C
*********************************************************************************************/

void main()
{

  ANSELC = 0;                          // Configure PORT C as digital
  TRISC = 0;                           // PORT C as output

  T2CON = 0b00000110;                  // Timer 2 with prescaler 16
  PR2 = 249;                           // Load PR2 register of Timer 2
  CCPTMRS0 = 0;                        // Enable PWM
  CCPR1L = 0X3E;                       // Load duty cycle
  CCP1CON = 0x2C;                      // Load duty cycle and enable PWM

  for(;;)                              // Wait here forever
  {
  }

}
```

Figure 9.51 Program listing using the PWM module

Substituting the values into the equation, and assuming a prescaler of 16, an oscillator frequency of 8 MHz (Tosc = 0.125 μs), the PWM period of 2000 μs, and duty cycle (ON time) of 500 μs, we get

$$PR2 = \frac{2 \times 10^{-3}}{16 * 4 * 0.125 \times 10^{-6}} - 1 \tag{9.11}$$

which gives PR2 = 249

The 10-bit value to be loaded into PWM registers are given by (see Chapter 2)

$$CCPR1L : CCP1CON < 5 : 4 >= \frac{PWM\ duty\ cycle}{TMR2PS * Tosc} \tag{9.12}$$

where the upper 8 bits will be loaded into register CCPR1L and the two LSB bits will be loaded into bits 4 and 5 of CCP1CON.

www.it-ebooks.info

or

$$CCPR1L : CCP1CON < 5 : 4 >= \frac{500 \times 10^{-6}}{16 \times 0.125 \times 10^{-6}} \tag{9.13}$$

which gives 250. This number in 10-bit binary is '00111110 10'. Therefore, the value to be loaded into bits 4 and 5 of CCP1CON are the two LSB bits. that is '10'. Bits 2 and 3 of CCP1CON must be set to HIGH for PWM operation, and bits 6 and 7 are not used. Therefore, CCP1CON must be set to ('X' is don't care):

```
XX101100 i.e. hexadecimal 0x2C
```

The number to be loaded into CCPR1L is the upper 8 bits (of 250), that is '00111110', that is hexadecimal $0 \times 3E$.

At the beginning of the program, PORT C is configured as digital output port. Then, Timer 2 is configured and timer register PR2 is loaded to give the required period. PWM registers CCPR1L and CCP1CON are loaded with the duty cycle (ON time) and the PWM module is enabled. The main program then waits forever, where the PWM works in the background to generate the required waveform.

9.8.10 Changing the LED Brightness using the PWM Module

The program given in Figure 9.51 generates a single PWM waveform to reduce the brightness of an LED. In this section we will modify the program, such that the LED brightness changes continuously, for example every second.

In reference to the duty cycle formula, we can write the duty cycle as:

$$PWM\ duty\ cycle = TMR2PS * Tosc * (CCPR1L : CCP1CON < 5 : 4 >)$$

or

$$PWM\ duty\ cycle = 16 * 0.125 * (CCPR1L : CCP1CON < 5 : 4 >)$$

giving

$$PWM\ duty\ cycle = 2 * (CCPR1L : CCP1CON < 5 : 4 >)\ in\ \mu s$$

Now, by loading different values into the register pair CCPR1L and CCP1CON, we can obtain different duty cycles and hence different LED brightness. Remembering that the period of our PWM wave is 2000 μs, we can load values from 0 to 1000 into the register pair to correspond with no brightness to full brightness. Let us assume that we wish to change the brightness in 10 steps. Then, we could increase the value loaded into these registers by 100 every second, so that the duty cycle changes by 200 μs. What we will see is that the brightness of the LED will change from 0 to full brightness in 10 seconds, in steps of 10 levels.

Figure 9.52 shows the modified program (LED16.C). Here, variable k is used to increment the 10-bit value of the register pair by 100, that is the duty cycle is incremented by 200 μs. Variable k is shifted right by 2 bits and the upper 8 bits of the 10-bit duty cycle is loaded into

www.it-ebooks.info

```
/*************************************************************************************************
                        CHANGE LED BRIGHTNESS USING PWM
                        ---------------------------------------------------------

In this project an LED is connected to pin CCP1 of a PIC18F45K22  microcontroller. A PWM
wave is generated with a period of 2,000us and variable duty cycle, using the PWM module
of the microcontroller. The duty cycle is varied from 0us to 2000us (i.e. full brightness). The
duty cycle is 10-bit variable stored in k. The upper 8-bits are copied into CCPR1L and the two
LSB bits are copied to bits 4 and 5 of CCP1CON

The microcontroller is operated with an 8MHz crystal.


Author:  Dogan Ibrahim
Date:    November, 2011
File:    LED16.C
*************************************************************************************************/

void main()
{
  unsigned int j, k = 0;

  ANSELC = 0;                          // Configure PORT C as digital
  TRISC = 0;                           // PORT C as output

  T2CON = 0b00000110;                  // Timer 2 with prescaler 16
  PR2 = 249;                           // Load PR2
  CCPTMRS0 = 0;

  for(;;)                              // DO FOREVER
  {
   Delay_Ms(1000);                     // Wait 1 second
   k = k + 100;                        // Increment k
   if(k > 1000)k = 0;                  // If end of duty cycle
   CCPR1L = k >> 2;                    //  Load CCPR1L with upper 8 bits
   J = k & 0x03;                       // Get two LSB bits
   J = j << 4;                         // Move to bit positions 4 and 5
   CCP1CON = 0x0C | j;                 // Load CCP1CON bits 4 and 5
  }

}
```

Figure 9.52 Modified program

register CCPR1L. Then, the two LSB bits of the duty cycle are extracted and ORed with register CCP1CON to enable the PWM and also to provide the 2 LSB bits of the duty cycle.

9.9 PROJECT 9.9 – LED Candle

9.9.1 Project Description

In this project we design an LED candle – an LED that imitates a burning real candle. The operation of the project is such that a PWM waveform is sent to the LED with random duty

www.it-ebooks.info

cycle. Because of this randomness, the LED lights up and flickers such that it looks as if it is a real candle.

9.9.2 Block Diagram

The block diagram of the project is as shown in Figure 9.3.

9.9.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 9.4, except that the LED is connected to port pin CCP1 (pin RC2 in PIC18F45K22 microcontroller) via a current limiting resistor.

9.9.4 Project PDL

The PDL of this project is given in Figure 9.53.

9.9.5 Project Program

The program is named LED17.C and the program listing of the project is shown in Figure 9.54. At the beginning of the main program, PORT C has been configured as digital output and Timer 2 and PWM modules have been configured to generate a PWM waveform on pin CCP1 (RC2) with a period of 2000 μs. A pseudorandom number generator function has been created.

```
BEGIN
        Configure PORT C
        Configure Timer 2 for PWM
        Configure PWM registers
        DO FOREVER
                Wait 500ms
                CALL Number to get a random number between 1 and 1000
                IF number < 50 THEN
                        Number = 50
                ENDIF
                Load PWM duty cycle registers CCPR1L and CCP1CON with this number
        ENDDO
END


BEGIN/NUMBER
        Generate a pseudorandom number
END/NUMBER
```

Figure 9.53 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************************
                                        LED CANDLE
                                    ----------------------

In this project an LED is connected to pin CCP1 of a PIC18F45K22 microcontroller. This
project imitates a real burning candle. A random Number is generated using a
pseudorandom Number generator. This Number is used to change the duty cycle of the
generated PWM waveform, thus giving the look of a real flickering burning candle.


Author:  Dogan Ibrahim
Date:    November, 2011
File:    LED17.C
*************************************************************************************************/
//
// Pseudo random number generator function
//
unsigned int Number(int Lim, int Y)
{
 unsigned int Result;
 static unsigned int Y;

 Y = (Y * 32719 + 3) % 32749;
 Result = ((Y % Lim) + 1);
 return result;
 }


//
// Main program
//
void main()
{
  unsigned int j,k = 0;
  unsigned char seed = 1;

  ANSELC = 0;                              // Configure PORTC as digital
  TRISC = 0;                               // PORT C as output

  T2CON = 0b00000110;                      // Timer 2 with prescaler 16
  PR2 = 249;                               // Load PR2
  CCPTMRS0 = 0;

  for(;;)                                  // DO FOREVER
  {
   Delay_Ms(500);                          // 500ms delay
   k = Number(1000, seed);                 // Get a random Number  1 to 1000
   if(k < 50)k = 50;                       // If less than 50, make it 50
   CCPR1L = k >> 2;                        // Load upper byte into CCPR1L
   j = k & 0x03;                           // Get two LSB bits
   j = j << 4;                             // Move to bit positions 4 and 5
   CCP1CON = 0x0C | j;                     // Load CCP1CON
  }
}
```

Figure 9.54 Program listing of the project

www.it-ebooks.info

The program operates in an endless loop created with a *for* statement. Inside this loop, a small delay is added between each iteration to slow down the rate of brightness changes. The random number generator function is called to generate a number between 1 and 1000, which corresponds to duty cycles between 2 and 2000 μs (see earlier project). This number is stored in variable *k*. In order to make sure that the LED does not turn OFF (i.e. the candle does not extinguish), the minimum value of *k* is set to 50, corresponding to a minimum duty cycle of 100 μs. PWM registers CCPR1L and CCP1CON are then loaded, as in the earlier project. The parameters given in this project can be adjusted to change the look and the flickering action of the LED as desired.

9.10 Summary

This chapter has given the design of LED based project using the PIC microcontrollers. All the projects given in the chapter have been tested and are fully working. The full circuit diagram and the source code are given for all the projects.

Exercises

9.1 An LED is connected to port pin RC0 of a PIC microcontroller. Write a program to flash the LED at a rate of 200 ms.

9.2 8 LEDs are connected to PORT C of a PIC microcontroller. Write a program to flash the odd numbered LEDs (at bit positions 1, 3, 5 and 7).

9.3 8 LEDs are connected to PORT C of a PIC microcontroller. In addition, a push-button switch is connected to port pin RB0. Write a program to turn ON the odd numbered LEDs (at bit positions 1, 3, 5 and 7) when the button is pressed and the even numbered LEDs (at bit positions 0, 2, 4 and 6) if the button is not pressed.

9.4 Design a microcontroller based two dice system. Assume that the dices are connected to PORT B and PORT C of the microcontroller. In addition, a push-button switch is connected to port pin RA0. When the button is pressed, two random dice numbers should be generated and the LEDs should indicate these dice numbers appropriately. The LEDs should remain on for 5 seconds and should then all go OFF to indicate that the system is ready to generate new dice numbers.

9.5 It is required to design an LED based Roulette game with 37 LEDs. Design the circuit diagram of the game. Describe how the LEDs can be controlled using only 7 I/O port pins.

9.6 Explain how the brightness of an LED can be changed by using a PWM signal to drive it.

9.7 Explain how the PWM module of a PIC microcontroller operates. Write a program to generate a PWM signal with an equal MARK to SPACE ratio, having a period of 1500 μs.

www.it-ebooks.info

10

7-Segment LED Display Based Projects

In this chapter we will look at the design of projects using 7-segment display devices. Initially, a simple 1-digit project is given to familiarise the reader with the interfacing and use of 7-segment displays. Then, the use of multiplexed 2-digit and 4-digit 7-segment display projects are given for more useful and practical applications.

10.1 PROJECT 10.1 – Single Digit Up Counting 7-Segment LED Display

10.1.1 Project Description

This is perhaps the simplest 7-segment LED project we can have. In this project, a 7-segment LED display is connected to PORT D of a PIC microcontroller. The LED counts up from 0 to 9 and then back to 0 continuously, with a 1 second delay between each count.

10.1.2 Block Diagram

The block diagram of the project is shown in Figure 10.1.

10.1.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 10.2. A common-cathode type display is used in this project (see Chapter 5), where the cathode is connected to ground and a segment is turned ON by setting it to logic HIGH. The segments are driven using current limiting resistors. In this project, a PIC18F45K22 type microcontroller is used, but any other model with at least 7 I/O ports can also be used. An 8 MHz crystal is used to provide the clock signals. The microcontroller is Reset using an external push-button.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

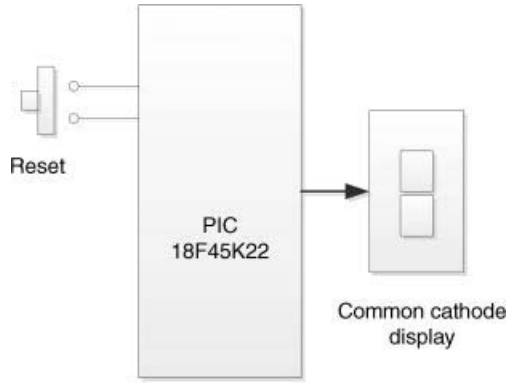


Figure 10.1 Block diagram of the project

10.1.4 Project PDL

The PDL of this project is very simple and is given in Figure 10.3.

10.1.5 Project Program

The program is named SEG1.C and the program listing of the project is given in Figure 10.4. At the beginning of the project, counter variable *Cnt* is initialised to zero. Also, the segment

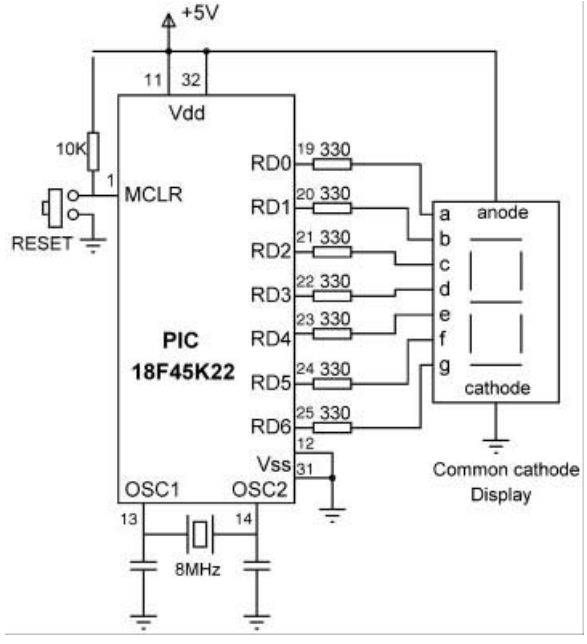


Figure 10.2 Circuit diagram of the project

www.it-ebooks.info

```
BEGIN
    Initialise Count to 0
    Store segment display data in an array (Table 5.3)
    Configure PORT D as digital and output
    DO FOREVER
        Get pattern corresponding to Count
        Send pattern to PORT D
        Increment Count
        IF Count is 10, reset to 0
        Wait 1second
    ENDDO
END
```

Figure 10.3 PDL of the project

patterns (see Table 5.3) for numbers 0 to 9 are stored in array *SevenSegment*. PORT D is configured as a digital I/O port by clearing register ANSELD (different PIC microcontrollers may require different settings). An endless loop is formed using a *for* statement. Inside this loop array, *SevenSegment* is indexed using the *Cnt* value. The corresponding bit pattern is stored in variable *Disp* and is then sent to PORT D to display the number in *Cnt*. Variable *Cnt* is incremented by one, a 1-second delay is introduced into the loop and the loop is repeated.

If you are using the EasyPIC 7 development board, then make sure you set to ON position switch SW4.1 (i.e. DIS0), to enable the rightmost digit of the 7-segment LED on the board. In addition, you will have to include the following statements before the *for* loop in Figure 10.4, to enable the display digit, by connecting the transistor connected to the common cathode pin. Notice that this transistor is connected to pin RA0 of the microcontroller (see Figure 10.5).

```
ANSELA=0;        // Configure PORT A as digital
TRISA=0;                 // Set PORT A pins as output
PORTA.F0=1;      // Enable common cathode of rightmost digit
```

10.1.6 Suggestions for Further Development

The program given in Figure 10.4 can be made more readable if the display part of the program can be combined in a function. Figure 10.6 shows the modified program (SEG2.C). Function *Display_Segment* receives a number between 0 and 9 and returns the bit pattern corresponding to this number. The main program then sends this bit pattern to PORT D to display the number on the 7-segment LED.

10.2 PROJECT 10.2 – Display a Number on 2-Digit 7-Segment LED Display

10.2.1 Project Description

This project shows how more than one 7-segment display can be multiplexed. In this project, we will display number 45 on a 2-digit display as an example.

www.it-ebooks.info

```
/*************************************************************************************************
                        1-DIGIT 7-SEGMENT LED COUNTER
                        =============================

In this project a common cathode 7-segmentdisplay is connected to PORT D of a
PIC18F45K22 type microcontroller (other types can also be used). The microcontroller
is operated from an 8MHz crystal.

The display counts from 0 to 9 and then back to 0 continuously with one second delay
between each count.

The connection between PORT D and the LED segments are as follows:

  LED segment        PORT D
     a                RD0
     b                RD1
     c                RD2
     d                RD3
     e                RD4
     f                RD5
     g                RD6


Author:  Dogan Ibrahim
Date:    November, 2011
File:    SEG1.C
*************************************************************************************************/

void main()
{
  unsigned char Disp, Cnt = 0;
  unsigned char SevenSegment[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
                      0x07,0x7F,0x6F};

  ANSELD = 0;                                  // Configure PORT D as digital
  TRISD = 0;                                   // PORT D pins are outputs

  for(;;)                                      // DO FOREVER
  {
    Disp = SevenSegment[Cnt];                  // Get bits of number to be displayed
    PORTD = Disp;                              // Display number
    Cnt++;                                     // Incremet count
    if(Cnt == 10)Cnt = 0;                      // Back to zero
    Delay_Ms(1000);                            // Wait 1 second
  }
}
```

Figure 10.4 Program listing of the project

www.it-ebooks.info

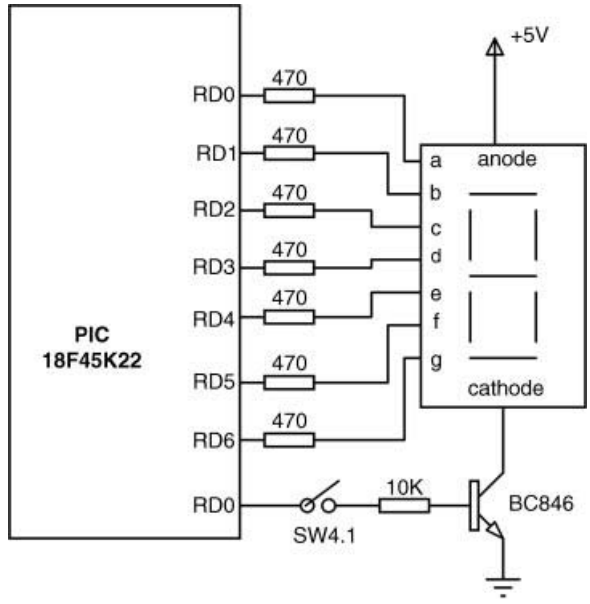


Figure 10.5 7-Segment LED connection on the Easy PIC 7 development board

10.2.2 Block Diagram

The block diagram of the project is shown in Figure 10.7.

10.2.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 10.8. The displays are connected in parallel to PORT D of the microcontroller. Each digit is enabled separately by connecting a transistor as a switch to its common cathode pin. Setting the base of the transistor to logic HIGH turns the transistor ON and enables the corresponding display digit. PORT A pins RA0 and RA1 are used to control the display enable lines. With multiplexed displays, each display is enabled for several milliseconds and the human eye thinks that all the LEDs are ON at all times. As an example, a two digit 7-segment display is operated as follows:

- send digit 1 number to PORT D;
- enable digit 1;
- wait for a few milliseconds;
- disable digit 1;
- send digit 2 number to PORT D;
- enable digit 2;
- wait for a few milliseconds;
- disable digit 2;
- repeat above steps forever.

www.it-ebooks.info

```
/*********************************************************************************************
                        1-DIGIT 7-SEGMENT LED COUNTER
                        =============================

In this project a common cathode 7-segment display is connected to PORT D of a
PIC18F45K22 type microcontroller (other types can also be used). The microcontroller is
operated from an 8MHz crystal.

The display counts from 0 to 9 and then back to 0 continuously with one second delay
between each count.

The connection between PORT D and the LED segments are as follows:

  LED segment          PORT D
     a                 RD0
     b                 RD1
     c                 RD2
     d                 RD3
     e                 RD4
     f                 RD5
     g                 RD6


This program uses a function to get the bit pattern for a given number. This bit pattern is
returned to the calling program.

Author:  Dogan Ibrahim
Date:      November, 2011
File:       SEG2.C
*********************************************************************************************/

//
// this function forms the bit pattern corresponding to a number between 0 and 9. This
// bit pattern is returned to the calling program
//
unsigned char Display_Segment(unsigned char Number)
{
  unsigned char SevenSegment[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
                        0x07,0x7F,0x6F};
  unsigned char Disp;

  Disp = SevenSegment[Number];              // Get bits of the number to be displayed
  return (Disp);                            // Return bits to main program
}


//
// Start of main program
//
void main()
{
```

Figure 10.6 Modified program

www.it-ebooks.info

```
  unsigned char Cnt = 0;

  ANSELD = 0;                          // Configure PORT D as digital
  TRISD = 0;                           // PORT D pins are outputs

  for(;;)                              // DO FOREVER
  {
    PORTD = Display_Segment(Cnt);      // Display number
    Cnt++;                             // Increment count
    if(Cnt== 10)Cnt = 0;               // Back to zero
    Delay_Ms(1000);                    // Wait 1 second
  }
}
```

Figure 10.6 (*Continued*)

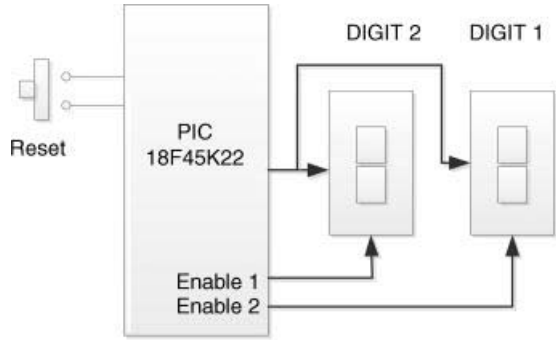


Figure 10.7 Block diagram of the project

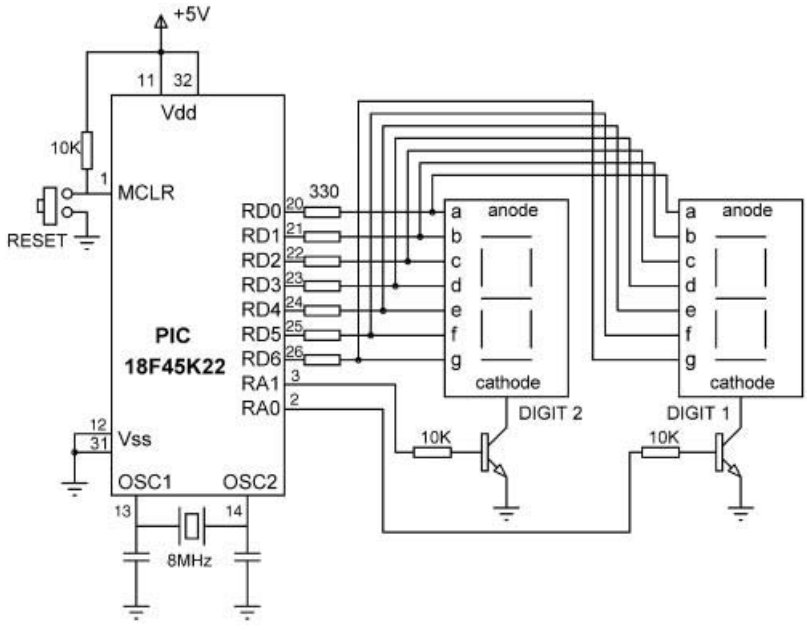


Figure 10.8 Circuit diagram of the project

www.it-ebooks.info

```
BEGIN
    Initialise variable Num to 45
    Configure PORT A and PORT D as digital output
    Disable both digits
    DO FOREVER
        Extract Digit 2 and Digit 1 data
        CALL Display_Segment to display 5
        Enable Digit 1
        Wait 5ms
        Disable Digit 1
        CALL Display_Segment to display 4
        Enable Digit 2
        Wait 5ms
        Disable Digit
    ENDDO
END


BEGIN/Display_Segment
    Create the 7-segment coding table
    Return bit pattern corresponding to the number
END/Display_Segment
```

Figure 10.9 PDL of the project

If you are using the EasyPIC 7 development board, then make sure you set to ON position switches SW4.1 (i.e. DIS0) and SW4.2 (i.e. DIS1), to enable the rightmost two digits of the 7-segment LEDs on the board.

10.2.4 Project PDL

The PDL of the project is given in Figure 10.9.

10.2.5 Project Program

The program is named SEG3.C and the program listing of the project is given in Figure 10.10. The 7-segment display function *Display_Segment* is also used in this program. At the beginning of the program, display enable signals RA0 and RA1 are assigned to symbols *Digit1_Enable* and *Digit2_Enable*, respectively. The number to be displayed (45) is stored in variable *Num*. PORT D and PORT Apins are configured as digital and output. Both display digits are disabled to start with. The program then enters an endless loop using the *for* statement. Inside this loop, the two digits of the number are extracted and stored in variables *Digit1_Data* and *Digit2_Data*. First, the bit pattern corresponding to *Digit1_Data* is obtained by calling function *Display_Segment* and this bit pattern is sent to PORT D to display number 5 on Digit 1. Digit 1 is enabled by sending logic 1 voltage to the base of the transistor connected to RA0.

www.it-ebooks.info

```
/*****************************************************************************************************
                                    2-DIGIT 7-SEGMENT DISPLAY
                                    =========================

In this project a 2-digit common cathode 7-segment display is connected to PORT D of a
PIC18F45K22 type microcontroller (other types can also be used). The microcontroller is
operated from an 8MHz crystal. The displays are multiplexed where the a-g segment
lines are in parallel, but the common cathode pin (enable) of each display is controlled
separately

Number 45 is displayed on the 2-digit display as an example.

The connection between PORT D and the LED segments are as follows:

  LED segment   PORT D
     a           RD0
     b           RD1
     c           RD2
     d           RD3
     e           RD4
     f           RD5
     g           RD6

Digit 1 (right hand side) is controlled from port RA0 through a transistor switch, and
Digit 2 (left hand side) is from port RA1.

This program uses a function to get the bit pattern for a given number. This bit pattern
is returned to the calling program.

Author:  Dogan Ibrahim
Date:    November, 2011
File:    SEG3.C
*****************************************************************************************************/
#define Digit1_Enable RA0_bit
#define Digit2_Enable RA1_bit

//
// this function forms the bit pattern corresponding to a number between
// 0 and 9. this bit pattern is returned to the calling program
//
unsigned char Display_Segment(unsigned char Number)
{
  unsigned char SevenSegment[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
                      0x07,0x7F,0x6F};
  unsigned char Disp;

  Disp = SevenSegment[Number];                  // Get bits of the number to be displayed
  return (Disp);                                // Return bits tomain program
}
```

Figure 10.10 Program listing of the project

www.it-ebooks.info

```
//
// Start of main program
//
void main()
{
  unsigned char Num = 45;
  unsigned char Digit1_Data,Digit2_Data;

  ANSELD = 0;                              // Configure PORT D as digital
  ANSELA = 0;                              // Configure PORT A as digital
  TRISD = 0;                               // PORT D pins are outputs
  TRISA = 0;                               // PORT A output
  Digit1_Enable = 0;                       // Disable Digit 1 to start with
  Digit2_Enable = 0;                       // Disable Digit 2 to start with

  for(;;)                                  // DO FOREVER
  {
    Digit2_Data = Num / 10;                // Extract Digit 2 data
    Digit1_Data = Num % 10;                // Extract Digit 1 data

    PORTD = Display_Segment(Digit1_Data);  // Send LSD number (5) to port
    Digit1_Enable = 1;                     // Enable Digit 1
    Delay_Ms(5);                           // Wait 5ms
    Digit1_Enable = 0;                     // Disable Digit 1
    PORTD = Display_Segment(Digit2_Data);  // Send MSD data (4) to port
    Digit2_Enable = 1;                     // Enable Digit 2
    Delay_Ms(5);                           // Wait 5ms
    Digit2_Enable = 0;                     // Disable Digit 2
  }
}
```

Figure 10.10 (*Continued*)

After a 5 ms delay, the number corresponding to Digit 2 is obtained and sent to PORT D. Digit 2 is enabled by setting RA1 to logic 1, and after a 5 ms delay, the digit is disabled. The above process is repeated forever, with a 5 ms delay after displaying each digit. The person looking at the digits thinks that both digits are ON at all times while number 45 is displayed.

It is important to realise that in Figure 10.10 the program is busy continuously refreshing the displays and the processor is not free to carry out any other tasks. We shall see in the next example how the display refreshing action can be implemented in a timer interrupt service routine so that the processor becomes free to do other tasks.

10.3 PROJECT 10.3 – Display Lottery Numbers on 2-Digit 7-Segment LED Display

10.3.1 Project Description

In this project, a 2-digit 7-segment display is connected to PORT D of the microcontroller. The project displays 7 random lottery numbers between 1 and 49 with a 5-second interval between each number. The last number displayed is considered to be the ‘bonus’. A push-

www.it-ebooks.info

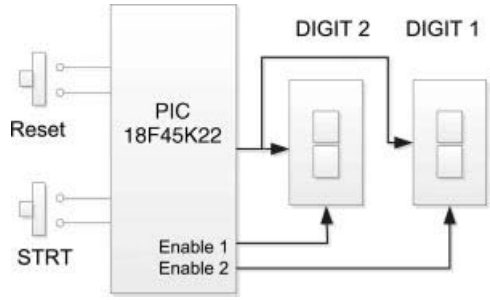


Figure 10.11 Block diagram of the project

button switch is used to start the game. At the end of displaying all the 7 numbers, the display goes blank to indicate that the circuit is ready to start a new game.

In this project, the displays are refreshed inside a timer interrupt service routine in the background, so that the processor is free to do other tasks.

10.3.2 Block Diagram

The block diagram of the project is shown in Figure 10.11. The 2-digit display pins a–g are connected in parallel and each digit is controlled separately. The push-button switch STRT starts the game.

10.3.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 10.12. The displays are connected in parallel to PORT D of the microcontroller. Each digit is enabled separately by connecting a transistor as a switch to its common cathode pin, as in the earlier project. Setting the base of the transistor to logic HIGH turns the transistor ON and enables the display. PORT A pins RA0 and RA1 are used to control the display enable lines. The STRT push-button switch is connected to pin RA2 of the microcontroller.

If you are using the EasyPIC 7 development board, then make sure you set to ON position switches SW4.1 (i.e. DIS0) and SW4.2 (i.e. DIS1), to enable the rightmost two digits of the 7-segment LED on the board.

10.3.4 Project PDL

The PDL of the project is given in Figure 10.13. The timer TMR0 is set to interrupt every 5 ms.

The interrupt priority feature is enabled by setting IPEN = 1 and the timer TMR0 interrupts are set to high-priority by setting TMR0IP = 1. The TMR0 interrupt flag is cleared by setting TMR0IF = 0, and the timer is set to operate in 8-bit mode, with a prescaler value of 64. As described in Chapter 2, the required timer value to generate interrupts every 5 ms (5000 μs) can be calculated from

$$\text{TMR0L} = 256 - 5000/(4 * \text{Tosc} * \text{Prescaler}) \qquad (10.1)$$

www.it-ebooks.info

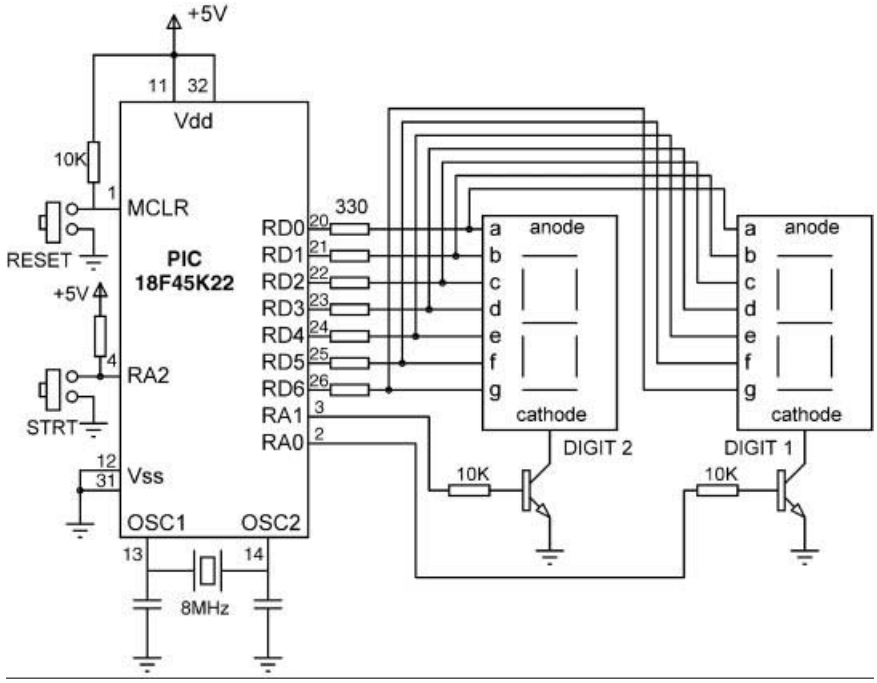


Figure 10.12 Circuit diagram of the project

or

$$\text{TMR0L} = 256 - 5000/(0.5 * 64) \tag{10.2}$$

giving TMR0L = 99.75; the nearest integer is selected, that is TMR0L = 100.

TMR0 interrupts are enabled by setting TMR0IE = 1. Finally, global interrupts are enabled by setting GIEH = 1. The timer is now ready to generate interrupts every 5 ms.

10.3.5 Project Program

The program is named SEG4.C and the program listing of the project is given in Figure 10.14. At the beginning of the main program, *Digit1_Enable*, *Digit2_Enable* and *STRT* symbols are defined as bits RA0, RA1 and RA2 of PORT A, respectively. PORT D and PORT A are configured as digital I/O, PORT D pins, and RA0, RA1 of PORT A are configured as outputs, while RA2 is configure as input.

The two digits are disabled by clearing *Digit1_Enable* and *Digit2_Enable* to start with, so that the displays are blank at the beginning of the program.

The program then configures TMR0 to generate interrupts every 5 ms. IPEN is set to 1 to enable priority based interrupts, and TMR0 is set to high priority by setting TMR0IP bit to 1. TMR0 is configured to operate as an 8-bit timer with a prescaler of 64 and TMR0L is loaded with 100 in order to overflow at 5 ms intervals. TMR0 interrupts are enabled by setting bit TMR0IE bit. Global interrupts are not enabled (GIEH = 0) at this point in the program.

www.it-ebooks.info

```
BEGIN
        Configure PORT a and PORT D
        Disable display digits
        Configure TMR0 for 5ms interrupts
        Disable interrupts
Next-game:
        Wait for STRT button to be pressed
        DO FOREVER
                CALL Number to get a random number between 1 and 49
                Check for no duplicates
                Enable interrupts
                Wait 5 seconds
                IF 7 numbers displayed THEN
                        Disable interrupts
                        Disable display digits
                        GOTO Next-game
                ENDIF
        ENDDO
END


BEGIN/NUMBER
        Generate a random number
        Return the number to the caller
END/NUMBER

BEGIN/DISPLAY_SEGMENT
        Load bit patterns for numbers 0 to 9 to an array
        Get the bit pattern of the required number
        Return the bit pattern to the caller
END/DISPLAY_SEGMENT

BEGIN/INTERRUPT
        Re-load timer register
        Extract digits of the number to be displayed
        IF Digit 1 is disabled THEN
                Disable Digit 2
                Send bit pattern to display Digit 1 data
                Enable Digit 1
        ELSE
                Disable Digit 1
                Supress leading zero
                Send bit pattern to display Digit 2 data
                Enable digit 2
        ENDIF
        Enable timer interrupt flag
END/INTERRUPT
```

Figure 10.13 PDL of the project

The program then waits until the *STRT* button is pressed. Normally this input pin is at logic 1 and pressing the button forces the RA2 pin to go to logic 0. When the button is pressed, the program clears all entries of an array called *Same*. This array is used to determine if duplicate numbers are generated and, if so, a new random number is generated.

www.it-ebooks.info

```
/*************************************************************************************************
        2-DIGIT 7-SEGMENT DISPLAY LOTTERY NUMBER GENERATOR
        ================================================

In this project a 2-digit common cathode 7-segment display is connected to PORT D of a
PIC18F45K22 type microcontroller (other types can also be used). The microcontroller is
operated from an 8MHz crystal. The displays are multiplexed where the a-g segment lines
are in parallel, but the common cathode pin (enable) of each display is controlled separately

The project displays the lottery numbers. A pseudorandom Number generator is used to
generate numbers between 1 and 49. 7 numbers are displayed with 5 second interval
between each Number. At the end the display goes blank to indicate that a new set of
7 numbers can be generated.

The game starts when the STRT button, connected to port RA2 is pressed.

The program uses timer interrupts evey 5ms to refresh the 7-segment displays.

The connection between PORT D and the LED segments are as follows:

  LED segment   PORT D
     a            RD0
     b            RD1
     c            RD2
     d            RD3
     e            RD4
     f            RD5
     g            RD6

Digit 1 (right hand side) is controlled from port RA0 through a transistor switch, and
Digit 2 (left hand side) is from port RA1. The STRT button is connected to pin RA2.
Normally this pin is at logic 1 and goes to logic 0 when the button is pressed.

This program uses a function to get the bit pattern for a given number. This bit pattern
is returned to the calling program.

Author:  Dogan Ibrahim
Date:    November, 2011
File:    SEG4.C
*************************************************************************************************/
#define Digit1_Enable RA0_bit
#define Digit2_Enable RA1_bit
#define STRT RA2_bit

unsigned char seed = 1;
unsigned char Num;

//
// this function forms the bit pattern corresponding to a number between 0 and 9.
// This bit pattern is returned to the calling program
//
```

Figure 10.14 Program listing of the project

www.it-ebooks.info

```
unsigned char Display_Segment(unsigned char Number)
{
  unsigned char SevenSegment[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
                    0x07,0x7F,0x6F};
  unsigned char Disp;

  Disp = SevenSegment[Number];                  // Get bits of the number to be displayed
  return (Disp);                                // Return bits tomain program
}


//
// Pseudo random number generator function
//
unsigned char Number(int Lim, int Y)
{
 unsigned char Result;
 static unsigned int Y;

 Y = (Y * 32719 + 3) % 32749;
 Result = ((Y % Lim) + 1);
 return Result;
}


// This is the Interrupt Service Routine (ISR). The program jumps to this routine every
// 1ms. Here, the display data is sent to the displays and the displays are refreshed
//
void interrupt(void)
{
  unsigned char Digit1_Data,Digit2_Data;

  TMR0L = 100;                                  // re-load TmR0L
  Digit2_Data = Num / 10;                       // Extract Digit 2 data
  Digit1_Data = Num % 10;                       // Extract Digit 1 data
  if(Digit1_Enable == 0)                        // If Digit 1 is not enabled already
  {
    Digit2_Enable = 0;
    PORTD = Display_Segment(Digit1_Data);       // Send LSD number (5) to port
    Digit1_Enable = 1;                          // Enable Digit 1
  }
  else
  {
    Digit1_Enable = 0;
    if(Digit2_Data != 0)                        // Supress leading 0
    {
      PORTD = Display_Segment(Digit2_Data);     // Send MSD data (4) to port
      Digit2_Enable = 1;                        // Enable Digit 2
    }
  }
  TMR0IF_bit = 0;                               // Clear TMR0 interrupt flag
```

Figure 10.14 (*Continued*)

www.it-ebooks.info

```
}


//
// Start of main program
//
void main()
{
  unsigned char Digit1_Data,Digit2_Data,Cnt,m,Flag,k;
  unsigned char Same[7];

  ANSELD = 0;                                   // Configure PORT D as digital
  ANSELA = 0;                                   // Configure PORT A as digital
  TRISD = 0;                                    // PORT D pins are outputs
  TRISA = 0x04;                                 // RA0,RA1 input, RA2 output
  Digit1_Enable = 0;                            // Disable Digit 1 to start with
  Digit2_Enable = 0;                            // Disable Digit 2 to start with
//
// Configure TMR0 to generate interrupts at every 5ms
//
  IPEN_bit = 1;                                 // Enable priority based interrupts
  TMR0IP_bit = 1;                               // TMR0 in high priority
  TMR0IF_bit = 0;                               // Clear TMR0 interrupt flag
  TMR0L = 100;                                  // Load TMR0L
  T0CON = 0b11000101;                           // TMR0 in 8 bit mode, prescaler=64
  TMR0IE_bit = 1;                               // Enable TMR0 interrupt
  GIEH_bit = 0;                                 // Disable global interrupts for now

nxt_game:                                       // Start of a new game
  while(STRT);                                  // Wait until STRT is pressed
   for(m = 0; m < 7; m++)Same[m] = 0;           // Clear duplicates array
   Cnt = 0;                                     //  Set no of generated numbers to 0
   k = 0;                                       // Array same index

   for(;;)                                      // DO FOREVER
   {
      do                                        // Check for duplicate numbers
      {
        Flag = 0;
        Num = Number(49, seed);                 // Get a random Number  1 to 49
        for(m = 0; m < 7; m++)if(Num == Same[m])Flag = 1;
      }while(Flag == 1);

      Same[k] = Num;                            // Save accepted number in array Same
      k++;
      GIEH_bit = 1;                             // Enable global interrupts
      Delay_Ms(5000);                           // Display for 5 seconds
      Cnt++;                                    // Increment no of displayed nos
      if(Cnt == 7)                              // If 7 numbers displayed
      {
```

Figure 10.14 (*Continued*)

www.it-ebooks.info

```
            GIEH_bit = 0;                    // Disable global interrupts
            Digit1_Enable = 0;               // Disable digit 1
            Digit2_Enable = 0;               // Disable digit 2
            goto nxt_game;                   // Back to start for a new game
          }
        }
    }
```

Figure 10.14 (*Continued*)

The main part of the program starts in a *for* loop. Inside this loop, a new random lottery number is generated between 1 and 49 by calling function *Number*. The program then checks to make sure that this number was not generated before. Variable *Flag* is set if the number is duplicated and, if so, a new number is generated.

The program then enables global interrupts by setting bit GIEH of INTCON. The newly generated number is displayed for 5 seconds. At the end of this period, the program checks to see whether or not 7 numbers have been generated, and if not, the program continues inside the *for* loop to generate another number. If, on the other hand, 7 numbers have been generated so far, then interrupts are disabled (GIEH = 0) and the two display digits are also disabled to indicate the end of the game. At this point, the program jumps to label *nxt_game* and waits until the *STRT* button is pressed again to start a new game.

Function *Display_Segment* returns the bit pattern corresponding to a number to be displayed.

Function *Number* generates a pseudorandom number between 1 and 49 (a lottery number) every time it is called. The seed for the number is set outside the program loop, so that a different set of numbers are generated when the *STRT* button is pressed.

The timer interrupt service routine (ISR) is identified by a function called *interrupt*. Here, the timer register TMR0L is re-loaded with 100. Then the digits of the generated number *Num* are extracted and stored in variables *Digit2_Data* and *Digit1_Data*. Inside the ISR, only one of the display digits is enabled. For example, if, on the last entry to the ISR, Digit 1 was enabled, then on the next, the ISR Digit 2 will be enabled and so on. The program also makes sure that a leading zero is supressed. If the Digit 2 data is 0, then this digit is disabled so that the leading zero is not shown.

Timer interrupt flag TMR0IF is re-enabled just before exiting the ISR, so that further timer interrupts can be accepted by the microcontroller.

10.4 PROJECT 10.4 – Event Counter Using 4-Digit 7-Segment LED Display

10.4.1 Project Description

In this project, a 4-digit 7-segment display is connected to PORT D of the microcontroller. The project counts external events and displays the count on the display; for example, counting the number of items passing on a conveyor belt in front of a sensor. Leading zeroes are suppressed in the display. A push button switch STRT is used to clear and start the count.

In this project, the displays are refreshed inside a timer interrupt service routine in the background, so that the processor is free to do other tasks.

www.it-ebooks.info

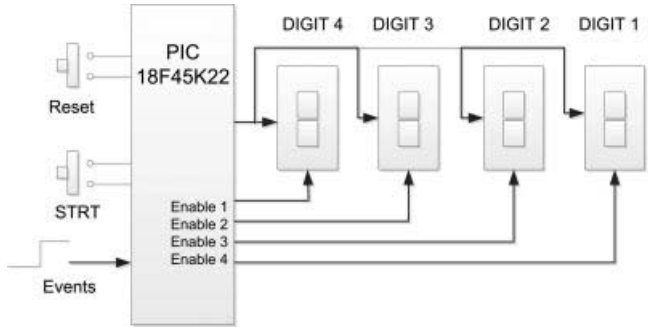


Figure 10.15 Block diagram of the project

10.4.2 Block Diagram

The block diagram of the project is shown in Figure 10.15. The 4-digit display pins a–g are connected in parallel and each digit is controlled separately. Counting starts when button STRT is pressed.

10.4.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 10.16. The displays are connected in parallel to PORT D of the microcontroller. Each digit is enabled separately by connecting a transistor as a switch to its common cathode pin, as in the earlier project. Setting the base of the transistor to logic HIGH turns the transistor ON and enables the display. PORT A pins RA0 to RA3 are used to control the display enable lines. The STRT push-button switch is connected to pin RA4 of the microcontroller. Normally RA4 pin is at logic 1 and pressing

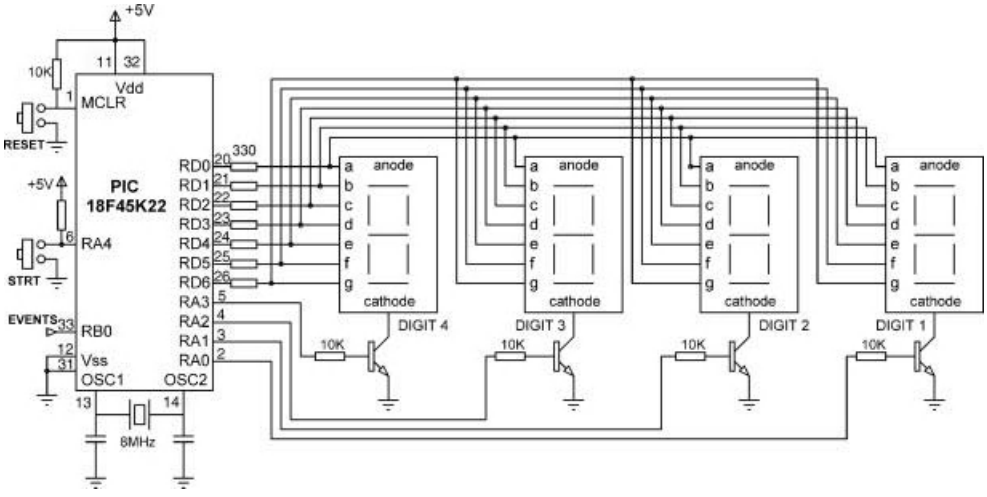


Figure 10.16 Circuit diagram of the project

www.it-ebooks.info

```
BEGIN
        Configure PORT A, PORT B and PORT Das digital output
        Disable display digits
        Configure TMR0 for 5ms interrupts
        Disable interrupts
        Wait for STRT button to be pressed
        Enable interrupts
        DO FOREVER
                IF event is detected THEN
                        Increment the count
                ENDIF
        ENDDO
END


BEGIN/DISPLAY_SEGMENT
        Load bit patterns for numbers 0 to 9 to a Table
        Get the bit pattern of the required number
        Return the bit pattern to the caller
END/DISPLAY_SEGMENT


BEGIN/INTERRUPT
        Re-load timer register
        IF Digit is 1 THEN
                CALL Display_Segment to display Digit 1
                Enable Digit 1
        ELSE IF Digit is 2 THEN
                CALL Display_Segment to display Digit 2
                Enable Digit 2
        ELSE IF Digit is 3 THEN
                CALL Display_Segment to display Digit 3
        ELSE IF Digit is 4 THEN
                CALL Display_Segment to display Digit 4
        ENDIF
        Clear timer interrupt flag
END/INTERRUPT
```

Figure 10.17 PDL of the project

the STRT button forces this pin to logic 0. Events are assumed to cause logic LOW-to-HIGH state changes on port pin RB0.

If you are using the EasyPIC 7 development board, then make sure that you set to ON position switches SW4.1 to SW4.4 (i.e. DIS0 to DIS3), to enable all the four 7-segment LED display digits on the board.

10.4.4 Project PDL

The PDL of the project is given in Figure 10.17. The timer TMR0 is set to interrupt every 5 ms as before. The timer interrupt is configured, as in the previous project and is not repeated here.

www.it-ebooks.info

10.4.5 Project Program

The program is named SEG5.C and the program listing of the project is given in Figure 10.18. At the beginning of the program, symbols *STRT* and *Event* are defined as bits RA4 and RB0, respectively. Then, PORT A, PORTB and PORT D are configured as digital I/O ports. PORT D pins are configured as outputs, bits RA0 to RA3 of PORT A are configured as outputs, and bit RB0 of PORT B is configured as an input port.

The display is blanked by setting PORT A to 0 to clear the enable lines of all ports. The event count is stored in variable *Num* and this variable is cleared to 0. The timer interrupt is then configured to provide interrupts at every 5 ms. IPEN is set to 1 to enable priority based interrupts, and TMR0 is set to high-priority by setting TMR0IP bit to 1. TMR0 is configured to operate as an 8-bit timer with a prescaler of 64 and TMR0L is loaded with 100 in order to overflow at 5 ms intervals. TMR0 interrupts are enabled by setting bit TMR0IE bit. Global interrupts are not enabled (GIEH = 0) at this point in the program.

The program then waits until the *STRT* button is pressed. Normally this input pin is at logic 1 and pressing the button forces the RA4 pin to go to logic 0. When the button is pressed, the program enabled global interrupts by setting GIEH = 1, so that timer interrupts can be accepted by the microcontroller.

The program then enters an endless loop using a *for* statement. At this point, the displays show 0 as the count is zero. Inside the *for* loop, the program waits while the Event line is at logic 0. After the Event line goes from 0 to 1, the event counter variable *Num* is incremented by 1. The program then waits until the Event line goes back to its idle state (logic 0).

The 7-segment LED digits are refreshed in the background inside the ISR. The ISR first re-loads TMR0L, so that it continues counting while the other tasks are executed. Function *Convert* is called to extract the digits of the current value of the event counter. This function is called with the following parameters:

```
Num: current value of the event counter
4: number of digits used
Digit_Data: an output array that will store the extracted digits
```

As an example, if Num = 3218, then the following values will be loaded into array *Digit_Data*:

```
Digit_Data[0] = 3
Digit_Data[1] = 2
Digit_Data[2] = 1
Digit_Data[3] = 8
```

Similarly, for example, if Num = 46, then the following values will be loaded into array *Digit_Data*:

```
Digit_Data[0] = 0
Digit_Data[1] = 0
Digit_Data[2] = 4
Digit_Data[3] = 6
```

www.it-ebooks.info

```
/*************************************************************************************************
                4-DIGIT 7-SEGMENT DISPLAY EVENT COUNTER
                =======================================

In this project a 4-digit common cathode 7-segment display is connected to PORT D of a
PIC18F45K22 type microcontroller (other types can also be used).  The microcontroller is
operated from an 8MHz crystal. The displays are multiplexed where the a-g segment lines
are in parallel, but the common cathode pin (enable) of each display is controlled separately

The project counts events where an event is defined as the change of the state of pin RB0
from logic 0 to logic 1. For example, the project can be used to counts objects passing on
a conveyor belt where a sensor generates a 0 to 1 logic signal whenever an object is detected.

The counting starts when the STRT button is pressed.

Counted objects are displayed on the 4-digit 7-segment display with the leading zeroes
suppressed.

The program uses timer interrupts every 5ms to refresh the 7-segment displays.

The connection between PORT D and the LED segments are as follows:

  LED segment   PORT D
    a             RD0
    b             RD1
    c             RD2
    d             RD3
    e             RD4
    f             RD5
    g             RD6

The display digits are controlled as follows:

   Digit      PORT pin
     1          RA0
     2          RA1
     3          RA2
     4          RA3

The STRT button is connected to port RA4 of the microcontroller. Normally RA4 is at logic 1.
Pressing the button forces this pin to go to logic 0.

Events are assumed to occur on pin RB0 of the microcontroller.


Author:  Dogan Ibrahim
Date:    November, 2011
File:    SEG6.C
*************************************************************************************************/
#define STRT RA4_bit
#define Event RB0_bit
```

Figure 10.18 Program listing of the project

www.it-ebooks.info

```
unsigned int Num;
unsigned char Next_Digit = 1;
unsigned char Digit_Data[4];

//
// This function extracts the digits of number N and stores them in array D. For example,
// if N = 2348 then D[0]=2,D[1]=2,D[2]=4,D[3]=8. Similarly, if N=56 then D[0]=0,D[1]=0,
// D[2]=5,D[3]=6. digits is the number of digits of number N, D is an array declared
// in the calling program. This function uses integers where the maximum N is 65535.
// With 4-digit display numbers 1 - 9999 can be displayed. "int" can be changed to
// "long int" for bigger numbers and more digits. For example, using a 8-digit display
// the maximum number that can be displayed will be 99,999,999 and this will require
// "long int".
//
void Convert(unsigned int N, unsigned char digits, unsigned char *D)
{
  unsigned int R, power;
  unsigned char i, j, k;

  j = digits;
  for(i = 0; i < digits; i++)                          // Do for all digits
  {
   power = 1;
   for(k = 0; k < j-1; k++)power = power * 10;         // Find power of 10
   j--;
   *(D + i) = N / power;
   R = N % power;
   N = R;
   }
}


//
// This function forms the bit pattern corresponding to a number between 0 and 9.
// This bit pattern is returned to the calling program
//
unsigned char Display_Segment(unsigned char Number)
{
  unsigned char SevenSegment[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,
                      0x07,0x7F,0x6F};
  unsigned char Disp;

  Disp = SevenSegment[Number];                 // Get bits of the number to be displayed
  return (Disp);                               // Return bits to main program
}


// This is the Interrupt Service Routine (ISR). The program jumps to this routine every
// 1ms. Here, the display data is sent to the displays and the displays are refreshed.
```

Figure 10.18 (*Continued*)

www.it-ebooks.info

```
// A switch statement is used to display data and refresh the displays
//
void interrupt(void)
{
  TMR0L = 100;                                    // Re-load TmR0L
  Convert(Num, 4, Digit_Data);

  switch(Next_Digit)
  {
   case 1:                                        // DIGIT 1
      PORTD = Display_Segment(Digit_Data[3]);     // Send DIGIT 1 data
      PORTA = 0x01;                               // Enable DIGIT 1
      Next_Digit = 2;                             // Next is DIGIT 2
      break;
   case 2:
      if(Num > 9)                                 // Suppress leading 0
      {
        PORTD = Display_Segment(Digit_Data[2]);   // Send DIGIT 2 data
        PORTA = 0x02;                             // Enable DIGIT 2
      }
      Next_Digit = 3;                             // Next is DIGIT 3
      break;
   case 3:
      if(Num > 99)                                // Suppress leading 0
      {
        PORTD = Display_Segment(Digit_Data[1]);   // Send DIGIT 3 data
        PORTA = 0x04;                             // Enable DIGIT 3
      }
      Next_Digit = 4;                             // Next is DIGIT 4
      break;
   case 4:
      if(Num > 999)                               // Suppress leading 0
      {
        PORTD = Display_Segment(Digit_Data[0]);   // Send DIGIT 4 data
        PORTA = 0x08;                             // Enable DIGIT 4
      }
      Next_Digit = 1;                             // Next is DIGIT 1
   }

  TMR0IF_bit = 0;                                 // Clear TMR0 interrupt flag
}


//
// Start of main program
//
void main()
{

  ANSELD = 0;                                     // Configure PORT D as digital
  ANSELA = 0;                                     // Configure PORT A as digital
```

Figure 10.18 (*Continued*)

www.it-ebooks.info

```
  ANSELB = 0;                                   // Configure PORT B as digital
  TRISD = 0;                                    // PORT D pins are outputs
  TRISA = 0x10;                                 // RA0-RA3 output, RA4 input
  TRISB = 1;                                    // RB0 is input

  PORTA = 0;                                    // Disable digits
  Num = 0;                                      // Clear count to start with
//
// ConfigureTMR0 to generate interrupts at every 5ms
//
  IPEN_bit = 1;                                 // Enable priority based interrupts
  TMR0IP_bit = 1;                               // TMR0 in high priority
  TMR0IF_bit = 0;                               // Clear TMR0 interrupt flag
  TMR0L = 100;                                  // Load TMR0L
  T0CON = 0b11000101;                           // TMR0 in 8 bit mode, prescaler=64
  TMR0IE_bit = 1;                               // Enable TMR0 interrupt
  GIEH_bit = 0;                                 // Disable global interrupts for no

  while(STRT);                                  // Wait until STRT is pressed
  GIEH_bit = 1;                                 // Enable timer interrupts

  for(;;)                                       // DO FOREVER
  {
    while(Event == 0);                          // Wait while Event is 0
    Num++;                                      // 0 to 1 detected, increment Num
    while(Event == 1);                          // Wait while Event is 1
  }
}
```

Figure 10.18 (*Continued*)

Then, a switch statement is used to send data and to enable each digit of the display. Variable *Next_Digit* takes values between 1 and 4 and determines which digit will be ON next. For example, when Next_Digit = 1, data is sent to DIGIT 1 of the display and this digit is enabled by sending a logic 1 to the digit enable transistor. *Next_Digit* is then set to 2 so that on the next entry to the ISR, DIGIT 2 will be processed, and so on. Notice that any leading zeroes are suppressed before displaying a digit other than DIGIT 1. Thus, for example, number '45' is displayed as ' 45' and not as '0045'. Just before exiting the ISR, the timer interrupt flag TMR0IF is cleared, so that further timer interrupts can be accepted by the microcontroller.

10.5 PROJECT 10.5 – External Interrupt Based Event Counter Using 4-Digit 7-Segment LED Display with Serial Driver

10.5.1 Project Description

In this project, a display module known as the BM08M04N-R is used. This is a 4-digit 7-segment LED display manufactured by Nexus Machines Ltd. This is a family of 7-segment displays ranging in size from 8 to 38 mm and available in colours of red, green and yellow. The project counts events occurring on external interrupt pin RB0/INT0. An event is said to occur when the state of RB0/INT pin goes from logic 0 to logic 1. This project is similar to

www.it-ebooks.info

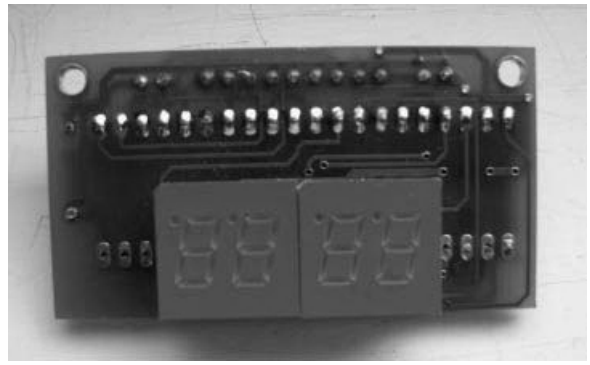


Figure 10.19 BM08M04N-R display

Project 10.4, but here the event is interrupt driven and also the display is a 4-digit 7-segment LED module with serial driver.

The BM08M04N-R display (see Figure 10.19) has an on-board controller chip that accepts data in serial format. The display has 9 pins, as shown in Table 10.1. 36 bits of serial data are sent to the display where a logic 1 turns a segment ON. The display has two pins (pin 1 and 2) where external LEDs can be connected via on-board current limiting resistors. The cathode of the external LEDs should be connected to these pins, and the anodes to the Vledpin (pin 9). The chip enable pin (pin 3) should be LOW for the display to be enabled. The brightness pin (pin 7) provides the segment brightness. A resistor is provided on the board that can be used, or an external resistor can be used to provide the required brightness. Serial clock pin (pin 5) clocks data into the display on the rising edge of the clock waveform. Thirty-six clocks are required to clock all the data in.

Table 10.2 shows how data should be sent to the display. The steps are summarised as follows:

- send Start bit (logic 1);
- send digits a1 to g1 of DIGIT 1 (rightmost digit);
- send decimal point (dp1) of DIGIT 1;
- send a2 to g2 of DIGIT 2;

Table 10.1 BM08M04N-R pin configuration

Pin no	Function
1	LED 1 drive
2	LED 2 drive
3	Chip enable
4	Data
5	Clock
6	Vdd (+5V)
7	Brightness
8	GND (0V)
9	Vled

www.it-ebooks.info

Table 10.2 Display data

Bit 0	Start	**Bit 9**	a2	**Bit 17**	a3	**Bit 25**	a4	**Bit 33**	LED 1
Bit 1	a1	**Bit 10**	b2	**Bit 18**	b3	**Bit 26**	b4	**Bit 34**	LED 2
Bit 2	b1	**Bit 11**	c2	**Bit 19**	c3	**Bit 27**	c4	**Bit 35**	Null
Bit 3	c1	**Bit 12**	d2	**Bit 20**	d3	**Bit 28**	d4		
Bit 4	d1	**Bit 13**	e2	**Bit 21**	e3	**Bit 29**	e4		
Bit 5	e1	**Bit 14**	f2	**Bit 22**	f3	**Bit 30**	f4		
Bit 6	f1	**Bit 15**	g2	**Bit 23**	g3	**Bit 31**	g4		
Bit 7	g1	**Bit 16**	dp2	**Bit 24**	dp3	**Bit 32**	dp4		
Bit 8	dp1								

- send dp2 of DIGIT 2;
- send a3 to g3 of DIGIT 3;
- send dp3 of DIGIT 3;
- send a4 to g4 of DIGIT 4 (leftmost digit);
- send dp4 of DIGIT 4;
- send LED1 bit;
- send LED2 bit;
- send a NULL bit.

The relationship between a number to be displayed and the bit pattern is given in Table 10.3. For example, to display number 5, we have to send 0 × B6 to the display. That is, the bit pattern '10110110'. The segment of each digit must be sent by shifting the bits to the left, that is the MSB bit is sent out first. Sending all zeroes to a digit blanks the digit and this is useful when we want to suppress leading zeroes.

As an example, suppose that we wish to display the number 3561. The steps should be as follows:

- Send a Start bit. That is send '1'.
- Send bit pattern for number 1 with no decimal point. That is, send '01100000'.

Table 10.3 Relationship between numbers and segment patterns

Number	a b c d e f g dp	Hexadecimal
0	1 1 1 1 1 1 0 0	0 × FC
1	0 1 1 0 0 0 0 0	0 × 60
2	1 1 0 1 1 0 1 0	0 × DA
3	1 1 1 1 0 0 1 0	0 × F2
4	0 1 1 0 0 1 1 0	0 × 66
5	1 0 1 1 0 1 1 0	0 × B6
6	1 0 1 1 1 1 1 0	0 × BE
7	1 1 1 0 0 0 0 0	0 × E0
8	1 1 1 1 1 1 1 0	0 × FE
9	1 1 1 1 0 1 1 0	0 × F6
Blank	0 0 0 0 0 0 0 0	0 × 00

www.it-ebooks.info

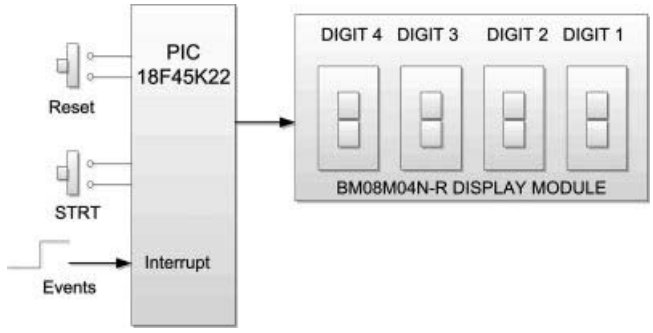


Figure 10.20 Block diagram of the project

- Send bit pattern for number 6 with no decimal point. That is, send '10111110'.
- Send bit pattern for number 5 with no decimal point. That is, send '10110110'.
- Send bit pattern for number 3 with no decimal point. That is, send '11110010'.
- Send zeroes for the two external LEDs. That is, send '00'.
- Send a NULL character. That is, send 0×0.

The event counting will start when the STRT button is pressed.

The advantage of using a display module, such as the BM08M04, is that all the display refreshing and control functions are done on the display board. We simply send the data to be displayed in serial format. As a result of this, the microcontroller is free to do other tasks while the data is displayed.

10.5.2 Block Diagram

The block diagram of the project is shown in Figure 10.20. Counting starts when the button STRT is pressed and events are recognised as external interrupts on external interrupt pin RB0/INT of the microcontroller.

10.5.3 Circuit Diagram

The circuit diagram of the project is simple and is shown in Figure 10.21. The STRT button is connected to pin RC0 of the microcontroller through a pull-up resistor. Normally RC0 is at logic 1 and goes to logic 0 when the button is pressed. Events are applied to the RB0/INT0 external interrupt pin of the microcontroller. The data and clock pins of the display are connected to port pins RC6 and RC7, respectively. Pin 6 of the display is connected to a +5 V power supply, and pins 3 and 8 are connected to ground.

10.5.4 Project PDL

The PDL of the project is given in Figure 10.22.

www.it-ebooks.info

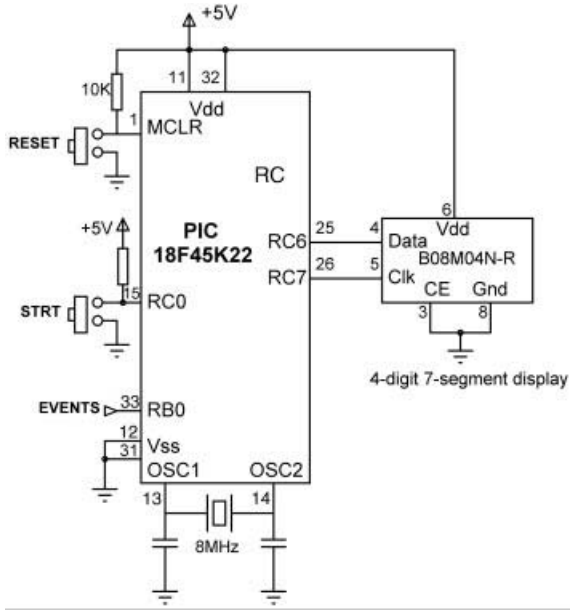


Figure 10.21 Circuit diagram of the project

10.5.5 Project Program

The program is named SEG6.C and the program listing of the project is given in Figure 10.23. At the beginning of the program, symbols *STRT*, *Data_Pin* and *Clk_Pin*are are defined as bits 0, 6 and 7 of PORT C, respectively.

In the main program, PORT B and PORT C are configured as digital ports. Port pins RB0 and RC0 are configured as inputs, while other bits of PORT B and PORT C are configured as outputs. The program then configures external interrupts on pin RB0/INT. IPEN is set to 1, to enable priority based interrupts. INT0IE is set to 1, to enable external interrupts on pin RB0/INT. Bit INTEDG0 is set to 1, so that external interrupts are accepted on the rising edge (logic 0 to 1) of the event signal. Global interrupts are not enabled at this stage (GIEH = 0). The program then waits until button *STRT* is pressed. When this button is pressed, the *STRT* pin goes to logic 0 and the program continues. Event counter is set to 0 × FFFF, so that the next count will be 0. An external interrupt is created by setting the interrupt flag INT0IF to 1 and by enabling global interrupts (GIEH = 1). As a result of this, event count will be incremented inside the ISR so that variable *Num* will become 0, and the display will show 0 to indicate that it is ready to start counting events.

Function *Convert* extracts the digits of the current value of the event counter. This function is called with the following parameters:

```
Num: current value of the event counter
4: number of digits used
Digit_Data: an output array that will store the extracted digits
```

www.it-ebooks.info

```
BEGIN
        Configure PORT B and PORT C
        Configure external interrupts on RB0/INT0 pin
        Wait until STRT button is pressed
        Display 0 to indicate that the event counter is ready
        Wait for external interrupts
END


BEGIN/CONVERT
        Extract digits of the number to be displayed
END/CONVERT


BEGIN/DISPLAY_SEGMENT
        Extract bit patterns of a digit
END/DISPLAY_SEGMENT


BEGIN/DISPLAY_NUMBERS
        Get bit patterns of all digits
        Suppress any leading 0s
        Load Start bit into an array
        Load segment data into an array
        Load LED1 and LED2 drive data into an array
        Load Null terminator into an array
        Send array data to the display with clock
END/DISPLAY_NUMBERS


BEGIN/INTERRUPT
        Increment event counter variable
        CALL Convert to extract digit numbers
        CALL Display_Numbers to display the event count
        Clear external interrupt flag
END/INTERRUPT
```

Figure 10.22 PDL of the project

As an example, if Num = 4597, then the following values will be loaded into array *Digit_Data*:

```
Digit_Data[0] = 4
Digit_Data[1] = 5
Digit_Data[2] = 9
Digit_Data[3] = 7
```

Function *Display_Segment* forms the bit pattern to be sent to a digit of the display, in order to display a number.

The ISR is defined by using reserved word interrupt and the program jumps here whenever an external event occurs. Inside the ISR, the event counter (*Num*) is incremented by 1. In addition, the digits of variable *Num* are extracted and stored in array *Digit_Data* by calling function *Convert*. Then, function *Display_Numbers* is called to display the total event count

www.it-ebooks.info

```
/*****************************************************************************************************
 EXTERNAL INTERRUPT BASED EVENT COUNTER USING 4-DIGIT 7-SEGMENT SERIAL
 DISPLAY MODULE
 ========================================================================
```

In this project a B08M04N-R type 4-digit 7-segment display with integral controller is used to display external events.

B08M04 devices are high efficiency red, green, and yellow colour 7-segment displays in sizes of 8mm to 38mm. In addition to 4-digit display, the device provides interface for two external LEDs.

Data is displayed by sending 36 bits of serial data that defines the segment data, decimal points, and the LED drive data. Sending a 1 turns ON that segment. Sending all 0s to a digit blanks that digit (used to suppress leading zeroes).

The advantage of using a display module such as the B08M04 is that all the display control and refreshing are done on the board and there is no need to refresh the displays externally. As a result of this the processor is free to do other tasks while the display is being refreshed. In addition, the display is controlled with only 2 pins (data and clock).

In this project the B08M04N-R display is connected to a PIC18F45K22 type microcontroller, operating with a 8MHz crystal. The connection between the display and the microcontroller I/O ports is as follows:

Display Pins	Microcontroller Pins
4 (Data)	RC6
5 (Clock)	RC7

The Start button is connected to port RC0 of the microcontroller. This pin is normally pulled HIGH using a pull-up resistor. Pressing the button forces a logic 0 at this pin.

External events are applied to the RB0/INT external interrupt pin of the microcontroller. An event is assumed to occur when this pin goes from logic 0 to logic 1 (i.e. on the rising edge of the event)

```
Author:   Dogan Ibrahim
Date:     November, 2011
File:     SEG6.C
*****************************************************************************************************/
#define STRT     RC0_bit
#define Data_Pin RC6_bit
#define Clk_Pin  RC7_bit

unsigned int Num;
unsigned char Digit_Data[4];

//
// This function extracts the digits of number N and stores them in array D. For example,
//  if N = 2348 then D[0]=2,D[1]=2,D[2]=4,D[3]=8. Similarly, if N=56 then D[0]=0,D[1]=0,
// D[2]=5,D[3]=6. digits is the number of digits of number N, D is an array declared in the
```

Figure 10.23 Program listing of the project

www.it-ebooks.info

```
// calling program.
//
// This function uses integers where the maximum N is 65535. With 4-digit display numbers
// 1 - 9999 can be displayed. "int" can be changed to "long int" for bigger numbers and more
// digits. For example, using a 8-digit display the maximum number that can be displayed
// will be 99,999,999 and this will require "long int".
//
void Convert(unsigned int N, unsigned char digits, unsigned char *D)
{
  unsigned int R, power;
  unsigned char i, j, k;

  j = digits;
  for(i = 0; i < digits; i++)                                    // Do for all digits
  {
   power = 1;
   for(k = 0; k < j-1; k++)power = power * 10;                   // Find power of 10
   j--;
   *(D + i) = N / power;
   R = N % power;
   N = R;
   }
}


//
// This function forms the bit pattern corresponding to a number between 0 and 9.
// This bit pattern is returned to the calling program
//
unsigned char Display_Segment(unsigned char Number)
{
  unsigned char SevenSegment[] = {0xFC,0x60,0xDA,0xF2,0x66,0xB6,0xBE,
                    0xE0,0xFE,0xF6};
  unsigned char Disp;

  Disp = SevenSegment[Number];                  // Get bits of the number to be displayed
  return (Disp);                                // Return bits to main program
}

//
// This function sends 36 bits to the display module to display the required number. Array D
// stores the digits to be displayed. For example if the number to be displayed is 3456 then
// D[0]=3,D[1]=4,D[2]=5,D[3]=6. Array DPattern stores the bit pattern to be sent to the
// display for each digit. Notice that DPattern[0] is the bit pattern for DIGIT 1 (rightmost
// digit), DPattern[1] is the bit pattern for DIGIT 2, DPattern[2] is the bit pattern for DIGIT 3,
// and DPattern[3] is the bit pattern for DIGIT 4 (leftmost digit).
//
// Leading zeroes are suppressed by sending all 0s to appropriate digit segments. Array
// DataArray stores all the 36 bits to be sent to the display, including the Start bit, digit
// segment bits, LED1 and LED2 drive bits, and the Null bit. After sending each bit the
// clock line is toggled.
```

Figure 10.23 (*Continued*)

www.it-ebooks.info

```
//
void Display_numbers(unsigned char D[])
{
  unsigned char DataArray[36];
  unsigned char i,m,j;
  unsigned char dp[] = {0,0,0,0};                    // No decimal points
  unsigned char DPattern[4];

  for(i=0; i<4; i++)DPattern[3-i] = Display_Segment(D[i]);    // Get digit bit patterns
//
// Suppress leading zeroes. Sending all 0s to a digit blanks that digit
//
  if(Num < 10)
  {
     Dpattern[1] = 0;
     Dpattern[2] = 0;
     Dpattern[3] = 0;
  }else if(Num < 100)
  {
     Dpattern[2] = 0;
     Dpattern[3] = 0;
  }
  else if(Num < 1000)Dpattern[3] = 0;

//
// Put all data in an array including the Start bit
//
  DataArray[0] = 1;                                  // Start bit

  for(i=0; i < 4; i++)                               // Do for all 4 digits
  {
    for(j=1; j<=7; j++)                              // Do for all 7 segments
    {
      m = DPattern[i] & 0x80;                        // Get MSB bit
      if(m != 0)DataArray[8*i+j] = 1; else DataArray[8*i+j]=0;
      DPattern[i] = DPattern[i] << 1;                // Shift left
    }
    DataArray[8*i+j+1] = dp[i];                      // Insert decimal point
  }

  DataArray[33] = 0;                                 // LED1 drive is 0
  DataArray[34] = 0;                                 // LED2 drive is 0
  DataArray[35] = 0x0;                               // Null terminator
//
// Now all the required 36 bits are in array DataArray. Send each bit to the display
// module, followed by a clock signal
//
  for(i=0; i < 36; i++)                              // Do for all 36 bits
  {
    Data_Pin = DataArray[i];                         // Send data bit
    Clk_Pin = 1;                                     // Clock = 1
```

Figure 10.23 (*Continued*)

www.it-ebooks.info

```
    Delay_us(1);                          // Wait to settle
    Clk_Pin = 0;                          // Clock = 0
    Delay_us(1);                          // Wait to settle
  }
}

// This is the Interrupt Service Routine (ISR). The program jumps to this routine every
// time an external interrupt occurs. i.e. everytime pin RB0/INT0 goes from logic 0 to
// logic 1. Here, the event count (in Num) is incremented, digits of the total events
// are extracted and the number is displayed on the display module.
//
void interrupt(void)
{
    Num++;                                // Increment event counter
    Convert(Num, 4, Digit_Data);          // Extract digits in array Digit_Data
    Display_Numbers(Digit_Data);          // Display the event number

    INT0IF_bit = 0;                       // Re-enable interrupt flag
}

//
// Start of main program
//
void main()
{
  ANSELB = 0;                             // Configure PORT B as digital
  ANSELC = 0;                             // Configure PORT C as digital
  TRISB = 1;                              // RB0 is input
  TRISC = 1;                              // Configure PORT C as output

  Clk_Pin = 0;                            // Clear clock pin to start with
//
// Configure RB0/INT0 external interrupt
//
  IPEN_bit = 1;                           // Enable priority based interrupts
  INT0IE_bit = 1;                         // Enable ext interrupts RB0/INT0
  INTEDG0_bit = 1;                        // Interrupt on 0 to 1
  INT0IF_bit = 0;                         // Clear RB0/INT0 interrupts
  GIEH_bit = 0;                           // Disable global interrupts

  while(STRT);                            // Wait until STRT is pressed

  Num = 0xFFFF;                           // Next number will be 0 when
                                          // interrupt occurs
  INT0IF_bit = 1;                         // Force external interrupt to
                                          // display 0 to start with
  GIEH_bit = 1;                           // Enable global interrupts

  for(;;)                                 // DO FOREVER
  {                                       // Wait for external interrupts
  }
}
```

Figure 10.23 (*Continued*)

www.it-ebooks.info

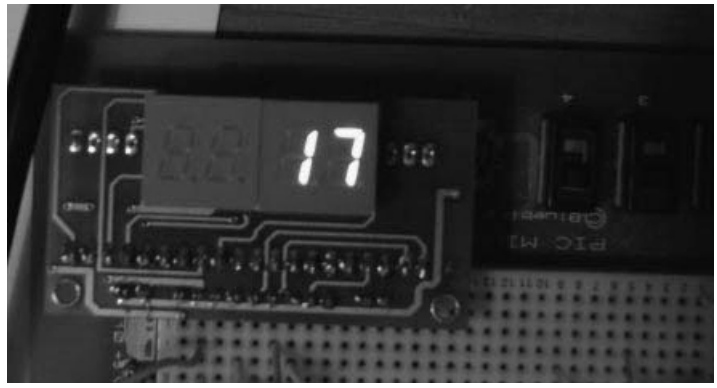


Figure 10.24 A number displayed on the display module

number on the display. The external interrupt flag for RB0/INT pin is cleared just before exiting the ISR, so that further external interrupts can be accepted by the microcontroller.

Function *Display_Numbers* is the most complicated function in the program. The decimal points (dp1, dp2, dp3 and dp4) are stored in an array called *dp*. As the decimal points are not used in this project, they are all cleared to 0. The bit patterns corresponding to each display digit are extracted by calling function *Display_Segment,* and are stored in array *DPattern*. Notice here that the elements of *DPattern* are reversed. That is *DPattern[0]* corresponds to DIGIT 1 (the rightmost digit). The function then checks for possible leading zeroes and sends all 0s to the digits that should be blanked. For example, if the event count is say 54, that is less that 100, then digits 2 and 3 should be blanked so that the display shows '54' and not '0054'. The next part of the function fills array *DataArray* with the 36-bit data to be sent to the display. Initially, the Start bit is loaded into *DataArray[0]*. Then the segment bits of each digit, starting from DIGIT 1, are shifted left by one and the MSB bit loaded into *DataArray*. This is done by logical ANDing the MSB bit with 0×80 to extract this bit, and then loading a 1 or a 0 into *DataArray*, depending on whether the MSB bit is set or not. After loading the segment bits of a digit, the decimal point of that digit is loaded, as described in Table 10.2. After loading all the necessary 36 bits, a *for* loop is formed to send each bit to the display, followed by a clock signal. A clock signal is defined as the 0 to 1 and then 1 to 0 transition of the display clock input. The manufacturer's data sheet specifies that the minimum clock HIGH and LOW times should be 950 ns. This is why a 1 μs delay is used after each edge of the clock signal.

Figure 10.24 shows a number displayed on the B08M04N-R display module.

10.6 Summary

This chapter has described the use of 7-segment LED displays in microcontroller based projects. At the beginning of the chapter, a project using a single-digit display is given. In later sections, the use of 2 and 4 digit multiplexed displays are described in projects. Finally, the operation of a 4-digit 7-segment display, with built-in serial driver, is described in a project to count external interrupt based events.

www.it-ebooks.info

Exercises

10.1 A 7-segment display is connected, as in Figure 10.2. Write a program to count the odd-numbered integers between 0 and 9 (i.e. 1,3,5,7 and 9) and display these numbers with 1 second intervals on the 7-segment display.

10.2 A 2-digit 7-segment display is connected, as in Figure 10.8. Write a program to count from 0 to 99 with 500 ms intervals and display the count on the 7-segment display.

10.3 Explain why a multiplexed 7-segment display should be refreshed in the timer interrupt service routine.

10.4 A 4-digit 7-segment display is connected, as in Figure 10.16. Write a program to count down from 1000 to 0, and display the count on the 7-segment display with 250 ms intervals.

10.5 A BM08M04N-R type 4-digit 7-segment display is connected to a microcontroller. Write a program to generate a random number between 1 and 1000 and display this number on the 7-segment display.

10.6 It is required to develop a dice program using a single digit 7-segment display. A push-button switch is connected to port pin RA0 of the microcontroller. Write a program to display a dice number between 1 and 6 on the display when the button is pressed. Describe how the program can be modified to display two dice numbers.

www.it-ebooks.info

11

Text Based LCD Projects

In this chapter we will look at the design of projects using text based LCDs. The LCDs used in the projects in this section are based on the HD44780 controller. The projects are organised by increasing difficulty and the reader is recommended to follow the projects in the given order.

11.1 PROJECT 11.1 – Displaying Text on LCD

11.1.1 Project Description

This is perhaps the simplest LCD project one can have. In this project, the text 'Hello' and 'LCD' are displayed on the first and second rows of an LCD, respectively. Text 'Hello' starts from column 1 and 'LCD' starts from column 5.

11.1.2 Block Diagram

The block diagram of the project is shown in Figure 11.1.

11.1.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 11.2. The LCD is connected to microcontroller PORT B pins, as in the EasyPIC 7 development board, that is the connection details are as follows:

LCD Pin	Microcontroller Pin
D4	RB0
D5	RB1
D6	RB2
D7	RB3
R/S	RB4
E	RB5

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

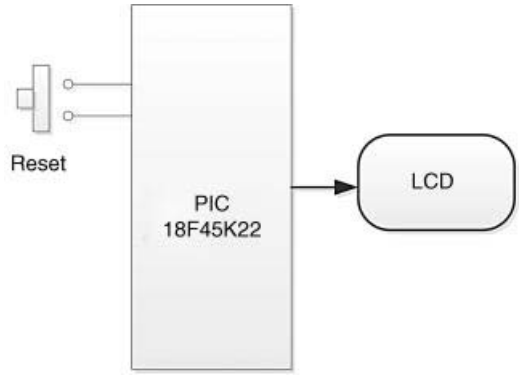


Figure 11.1 Block diagram of the project

The contrast of the LCD is controlled by connecting a 10 KB potentiometer to pin 3 of the LCD. A PIC18F45K22 type microcontroller is used with an 8 MHz crystal (any other type of PIC microcontroller can also be used if desired). The microcontroller is Reset using an external push-button.

If you are using the EasyPIC 7 development board, you can turn the LCD backlight ON by setting switch SW4.6 to ON position.

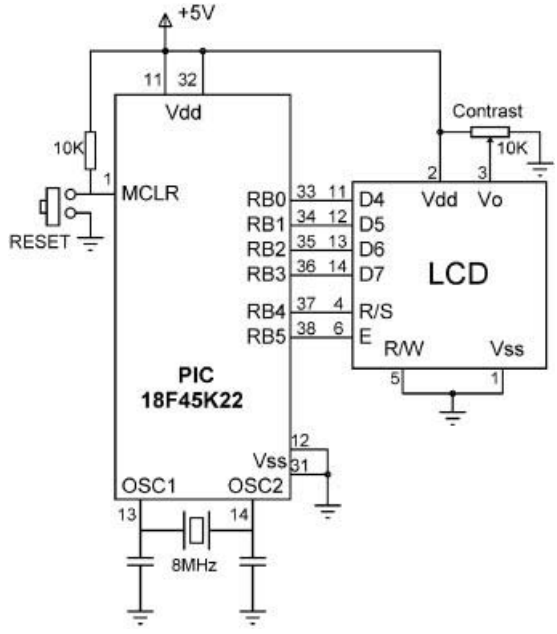


Figure 11.2 Circuit diagram of the project

www.it-ebooks.info

```
BEGIN
    Define the connection between the LCD and the microcontroller
    Configure PORT B as digital output
    Initialize LCD
    Clear LCD screen
    Display message on the LCD
END
```

Figure 11.3 PDL of the project

11.1.4 Project PDL

The PDL of this project is very simple and is given in Figure 11.3.

11.1.5 Project Program

The program is named LCD1.C and the program listing of the project is shown in Figure 11.4. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements. In the main program, PORT B is configured as a digital output port. The LCD library is initialised by calling the Lcd_Init function. Then the LCD is cleared and the cursor is disabled. Text *Hello* is displayed starting from row 1, column 1, by entering the text directly into function Lcd_Out. Text *LCD* is displayed starting from row 2, column 5, by declaring the text in a string (character array terminated with a Null character) named *Txt*. The program then waits in a loop forever. Notice that in microcontroller based applications, because there is no operating system to return to, the end of a program must be well defined. If the program is in a continuous loop, then there is no problem. But in examples such as here, after displaying the text, the program should be stopped by creating an endless loop.

Figure 11.5 shows the text displayed on the LCD.

11.2 PROJECT 11.2 – Moving Text on LCD

This project will show how we can move text on the LCD. Here, we will display a text on both rows of a 2×16 LCD and then move the text left and right. Initially, text 'Shift' will be displayed starting from row 1, column 1 of the LCD. Similarly, text 'LCD' will be displayed starting on row 2, column 5. The program will then shift the displayed texts 6 positions to the right, wait for 5 seconds, and then shift them left by 6 positions back to their original places. A 1 second delay will be introduced between each shift operation. The text positions on the LCD are shown below:

```
1234567890123456
Shift                          Initial text position
   LCD
     Shift                     Text shifted right by 6 positions
          LCD
Shift                          Text shifted left by 6 positions
    LCD
```

The delay between each shift operation can be adjusted, as required in the program.

www.it-ebooks.info

```
/*********************************************************************************************
                                    WRITE TEXT TO LCD
                                    =================

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC  microcontroller
can be used if desired).

The LCd is connected to PORT B of the microcontroller as follows:

 LCD pin   Microcontroller pin
   D4          RB0
   D5          RB1
   D6          RB2
   D7          RB3
   R/S         RB4
   E           RB5

R/W pin of the LCd is not used and is connected to GND. The brightness of the LCDis
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD.
Other pins of the potentiometer are connected to power and ground.

 The following text is displayed on the LCD:

    HELLO
      LCD

Author:    Dogan Ibrahim
Date:      November, 2011
File:      LCD1.C
*********************************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

unsigned char Txt[] = "LCD";

void main()
{
```

Figure 11.4 Program listing of the project

www.it-ebooks.info

```
    ANSELB = 0;                          // Configure PORT B as digital
    TRISB = 0;                           // Configure PORT B pins as output

    Lcd_Init();                          // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR);                 // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF);            // Cursor off
    Lcd_Out(1, 1, "Hello");              // Write text starting Row 1, Column 1
    Lcd_Out(2, 5, Txt);                  // write text starting Row 2, Column 5

    while(1);                            // End of program, wait here forever
}
```

Figure 11.4 (*Continued*)

Figure 11.5 Displaying text on the LCD

11.2.1 Block Diagram

The block diagram of the project is as in Figure 11.1.

11.2.2 Circuit Diagram

The circuit diagram of the project is as in Figure 11.2.

11.2.3 Project PDL

The PDL of this project is very simple and is given in Figure 11.6.

11.2.4 Project Program

The program is named LCD2.C and the program listing of the project is shown in Figure 11.7. At the beginning of the project, the connection between the microcontroller and the LCD are defined using *sbit* statements. In the main program, PORT B is configured as a digital output port. The LCD library is initialised by calling the Lcd_Init function. Then the LCD is cleared and the cursor is disabled. Text *Shift* is displayed starting from row 1, column 1, and text *LCD* is displayed starting from row 2, column 5.

The main part of the program is in an endless loop formed using a *for* statement. Inside this loop, the text is moved right by 6 positions by calling function *Move_Text_Right*. After a 5 second delay, the text is moved left by 6 positions by calling function *Move_Text_Left*. Both of these functions require the number of places to be shifted (*N*), and the delay between each shift operation (*Dly*) to be entered in arguments when the functions are called. The LCD

www.it-ebooks.info

```
BEGIN
        Define LCD to microcontroller connections
        Configure PORT B as digital and output
        Initialise LCD
        Clear LCD
        Turn OFF cursor
        Display text "Shift" at row 1, column 1
        Display text "LCD" at row 2, column 5
        DO FOREVER
                Shift display right by 6 positions
                Wait 5 seconds
                Shift display left by 6 positions
                Wait 5 seconds
        ENDDO
END


BEGIN/MOVE_TEXT_RIGHT
        DO required number of times
                Use command _LCD_SHIFT_RIGHT to shift display right
                Wait 1 second
        ENDDO
END/MOVE_TEXT_RIGHT


BEGIN/MOVE_TEXT_LEFT
        DO required number of times
                Use command _LCD_SHIFT_LEFT to shift display left
                Wait 1 second
        ENDDO
END/MOVE_TEXT_LEFT
```

Figure 11.6 PDL of the project

commands _LCD_SHIFT_RIGHT and _LCD_SHIFT_LEFT are used to shift the text right and left, respectively.

In the program given in Figure 11.7, the delay between each shift operation is specified as 1 second and the characters can be seen as they are being shifted. In practical applications, a much smaller delay (e.g. 10 ms or less) should be used, so that the shift operation is very quick.

11.3 PROJECT 11.3 – Counting with the LCD

In this project, we will see how to display numbers in addition to text on the LCD. Here, an up-counter will be designed that will count with 1 second intervals. The count will be displayed as follows:

```
Cnt = nnn
```

11.3.1 Block Diagram

The block diagram of the project is as in Figure 11.1.

www.it-ebooks.info

```
/*************************************************************************************************
                        MOVING TEXT ON LCD
                        ==================

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC  microcontroller
can be used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

 LCD pin    Microcontroller pin
   D4           RB0
   D5           RB1
   D6           RB2
   D7           RB3
   R/S          RB4
   E            RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD.
Other pins of the potentiometer are connected to power and ground.

In this project a text is displayed on both rows of the LCD. This text is then shifted left
and right with 5 second delay between each shift operation.

The following text is initially displayed on the LCD:

     Shift
       LCD

Then the text is shifted right by 6 positions. After a 5 second delay the text is shifted
6 positions to the left, back to its original position. This process is repeated forever.

Author:  Dogan Ibrahim
Date:      November, 2011
File:       LCD2.C
*************************************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
```

Figure 11.7 Program listing of the project

www.it-ebooks.info

```
// This function moves the display by "N" positions to the right.
// "Dly" milliseconds delay is used between each shift operation.
// For a quick shift, set "Dly" to around 10ms
//
void Move_Text_Right(unsigned char N, unsigned int Dly)
{
 unsigned char i;

 for(i = 0; i< N; i++)
 {
   Lcd_Cmd(_LCD_SHIFT_RIGHT);                    // Shift right
   VDelay_Ms(Dly);                               // Wait a bit
 }
}

//
// This function moves the display by "N" positions to the left.
// "Dly" milliseconds delay is used between each shift operation.
// For a quick shift, set "Dly" to around 10ms
//
void Move_Text_Left(unsigned char N, unsigned int Dly)
{
 unsigned char i;

 for(i = 0; i< N; i++)
  {
   Lcd_Cmd(_LCD_SHIFT_LEFT);                     // Shift left
   VDelay_Ms(Dly);                               // Wait a bit
  }
}

//
// Start of Main program
//
void main()
{
    ANSELB = 0;                                  // Configure PORT B as digital
    TRISB = 0;                                   // Configure PORT B pins as output

    Lcd_Init();                                  // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR);                         // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF);                    // Cursor off
    Lcd_Out(1, 1, "Shift");                      // Write text at Row 1, Column 1
    Lcd_Out(2, 5, "LCD");                        // Write text at Row2, Column 5

for(;;)                                          // DO FOREVER
    {
      Move_Text_Right(6,1000);                   // Move right by 6 places with 1 sec delay
      Delay_Ms(5000);                            // Wait 5 seconds
      Move_text_Left(6,1000);                    // Move left by 6 places with 1 sec delay
      Delay_Ms(5000);                            // Wait 5 seconds
   }
}
```

Figure 11.7 (*Continued*)

www.it-ebooks.info

```
BEGIN
    Define LCD to microcontroller connections
    Configure PORT B as digital and output
    Initialise count to zero
    Initialise LCD
    Clear LCD
    Turn OFF cursor
    DO FOREVER
        Increment count
        Convert count to string
        Display count
        Wait 1 second
    ENDDO
END
```

Figure 11.8 PDL of the project

11.3.2 Circuit Diagram

The circuit diagram of the project is as in Figure 11.2.

11.3.3 Project PDL

The PDL of this project is very simple and is given in Figure 11.8.

11.3.4 Project Program

The program is named LCD3.C and the program listing of the project is shown in Figure 11.9. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements and the counter variable, *Cnt*, is initialised to 0. In the main program, PORT B is configured as a digital output port. The LCD library is initialised, LCD is cleared, and the cursor is turned OFF. Then an endless loop is formed using a *for* statement. Inside this loop, the counter variable *Cnt* is incremented by one. This variable is then converted into a string using the mikroC Pro for PIC built-in library function ByteToStr. Here, the numeric variable is the first argument, and address of a character array, where the converted string will be stored, is the second argument. The character array must have a size of at least 4 bytes. The converted string equivalent of the count is stored, starting from address 5 of array *Txt* as follows:

```
Txt -> 0 1 2 3 4 5 6 7 8 9
       C n t = n nn
```

Thus, for example, number 1 is displayed as Cnt = 1, where there are two spaces before the actual number to be displayed. The program then waits for 1 second and the above process is repeated.

www.it-ebooks.info

```
/*************************************************************************************************
                                    COUNTING WITH THE LCD
                                    =====================

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC  microcontroller
can be used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

  LCD pin     Microcontroller pin
   D4           RB0
   D5           RB1
   D6           RB2
   D7           RB3
   R/S          RB4
   E            RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD. Other
pins of the potentiometer are connected to power and ground.

In this project an up counter is designed which displays numbers counting up every
second on the display. The display format is as follows:

      Cnt = nnn

Because an "unsigned char" is used to store the count, "nnn" is from 0 to 255.


Author:   Dogan Ibrahim
Date:     November, 2011
File:     LCD3.C
*************************************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections
```

Figure 11.9 Program listing of the project

www.it-ebooks.info

```
//
// Start of Main program
//
void main()
{
   unsigned char Cnt = 0;                          // Initialise Cnt to 0
   unsigned char Txt[] = "Cnt =    ";              // Display format

   ANSELB = 0;                                     // Configure PORT B as digital
   TRISB = 0;                                      // Configure PORT B pins as output

   Lcd_Init();                                     // Initialize LCD
   Lcd_Cmd(_LCD_CLEAR);                            // Clear display
   Lcd_Cmd(_LCD_CURSOR_OFF);                       // Cursor off

   for(;;)                                         // DO FOREVER
   {
     Cnt++;                                        // Increment Cnt
     ByteToStr(Cnt, Txt+5);                        // Convert Cnt to string
     Lcd_Out(1,1, Txt);                            // Display Txt (Cnt = nn)
     Delay_Ms(1000);                               // Wait 1 second
   }
}
```

Figure 11.9 (*Continued*)

11.3.5 Suggestions for Further Development

As shown above, the function ByteToStr inserts leading spaces when it converts a number into a string. In some applications, we may want to remove these spaces, and for example, display number 1 as:

```
Cnt =1
```

This can easily be done by using the library function Ltrim to remove leading spaces. The main part of the modified program listing is shown in Figure 11.10. Here, the count is converted into string and stored in character array pointed to by *Txt1*. Leading spaces are then removed with the Ltrim function and the new array is pointed to by *Txt2*. Library function strcat appends two strings, whose addresses are supplied as arguments. The string whose address is the second argument is appended to the string whose address is the first argument. The first string is not changed up to the point where the Null character is located, and the second string is appended, starting from this point. In Figure 11.10, the Null character is set at position 5 of the first string (i.e. at the point Cnt =), so that numbers are displayed with no leading zeroes.Variable *res* is a pointer to the appended string.

11.4 PROJECT 11.4 – Creating Custom Fonts on the LCD

There are some applications where we may want to create custom fonts, such as special characters, symbols or logos on the LCD. This project will show how to create the symbol of an arrow pointing upwards on the LCD, and then display 'Up arrow - <symbol of up arrow>' on the first row of the LCD.

www.it-ebooks.info

```
void main()
{
   unsigned char Cnt = 0;                    // Initialise Cnt to 0
   unsigned char Txt1[4];
   unsigned char *Txt2;
   unsigned char Disp[]="Cnt =   ";
   unsigned char *res;

   ANSELB = 0;                               // Configure PORT B as digital
   TRISB = 0;                                // Configure PORT B pins as output

   Lcd_Init();                               // Initialize LCD
   Lcd_Cmd(_LCD_CLEAR);                      // Clear display
   Lcd_Cmd(_LCD_CURSOR_OFF);                 // Cursor off

   for(;;)                                   // DO FOREVER
   {
     Lcd_Cmd(_LCD_CLEAR);
     Cnt++;                                  // Increment Cnt
     ByteToStr(Cnt, Txt1);                   // Convert Cnt to string
     Txt2=Ltrim(Txt1);                       // Remove leading spaces
     Disp[5]=0;                              // Get ready to append strings
     res=strcat(Disp,Txt2);                  // res is the appended string
     Lcd_Out(1,1, res);                      // Display as Cnt = nnn with no leading 0s
     Delay_Ms(1000);                         // Wait 1 second
   }
}
```

Figure 11.10 Modified program listing

mikroC Pro for PIC compiler provides a tool that makes the creation of custom fonts very easy. The steps for creating a font of any shape are given below:

- Start mikroC Pro for PIC compiler.
- Select Tools -> LCD Custom Character. You will see the LCD font editor form shown in Figure 11.11.
- Select 5 × 7 (the default).
- Click 'Clear all' to clear the font editor.
- Now, draw the shape of your font by clicking on the squares in the editor window. In this project, we will be creating the symbol of an 'up arrow', as shown in Figure 11.12.
- When you are happy with the font, click the 'mikroC Pro for PIC' tab so that the code generated will be for the mikroC Pro for PIC compiler.
- Click Generate Code' button. You will get the code, as shown in Figure 11.13.
- Click 'Copy Code To Clipboard' to save the code.
- We shall see later in the project how to display this font using the generated code.

11.4.1 Block Diagram

The block diagram of the project is as in Figure 11.1.

11.4.2 Circuit Diagram

The circuit diagram of the project is as in Figure 11.2.

www.it-ebooks.info

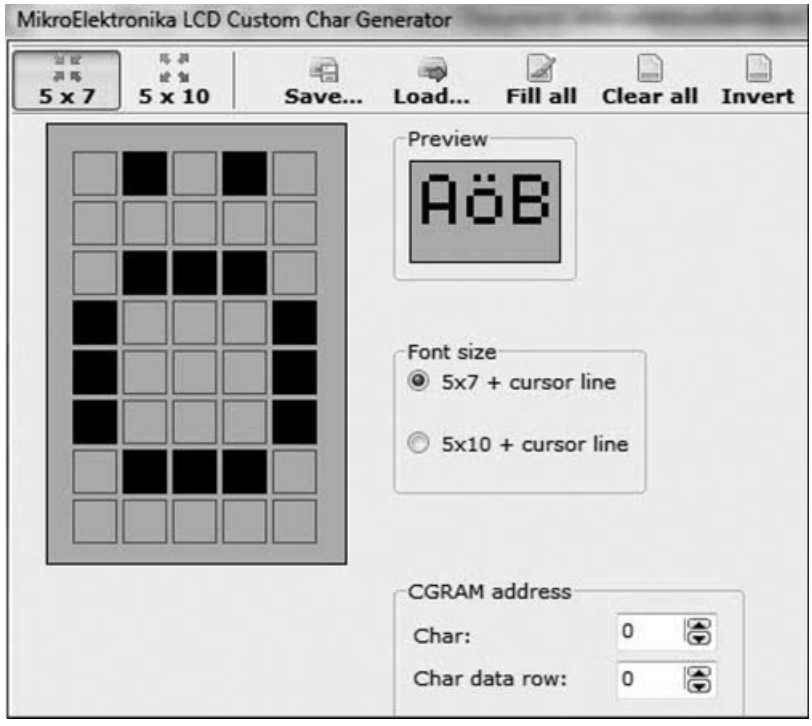


Figure 11.11 LCD font editor

11.4.3 Project PDL

The PDL of this project is very simple and is given in Figure 11.14.

11.4.4 Project Program

The program is named LCD4.C and the program listing of the project is shown in Figure 11.15. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements. PORT B is configured as a digital output port. The LCD is initialised, cleared and the cursor is turned OFF. Then the Lcd_Out function is called to display the text 'Up arrow – ', starting at row 1 and column 1 of the LCD. Function CustomChar is generated by the compiler and this function displays the created font at the specified row and column positions.

Figure 11.16 shows a picture of the LCD display.

11.5 PROJECT 11.5 – LCD Dice

In this project, two dice numbers are displayed on the LCD when the user presses a button called STRT. The text 'Good Luck' is displayed in the first row of the LCD. The two dice numbers are displayed in the second row in the following format (assuming the numbers are 4 and 6):

```
Good Luck
   4  6
```

www.it-ebooks.info

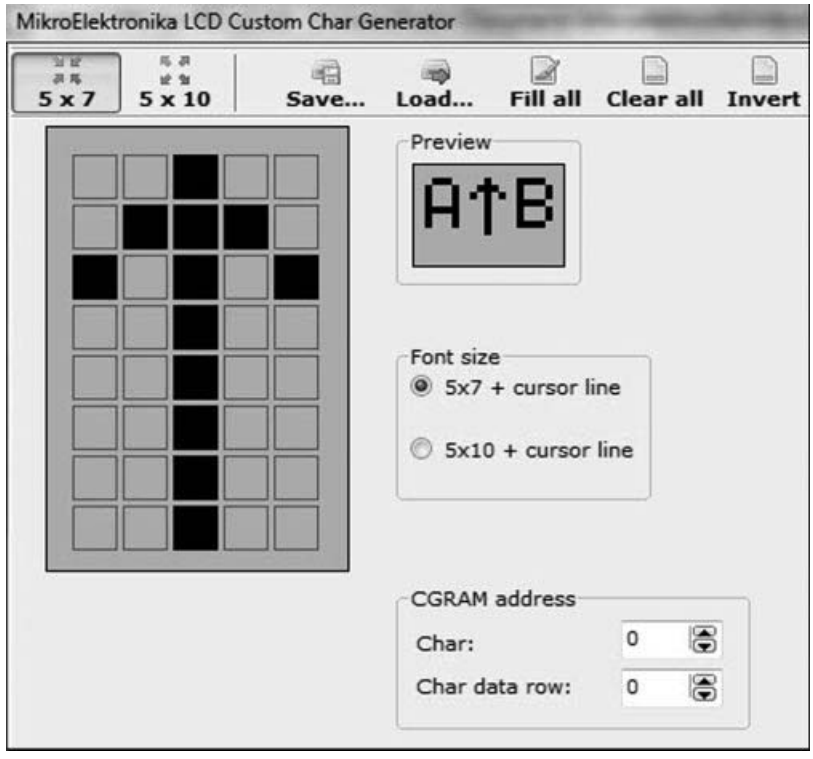


Figure 11.12 Creating an 'up arrow' font

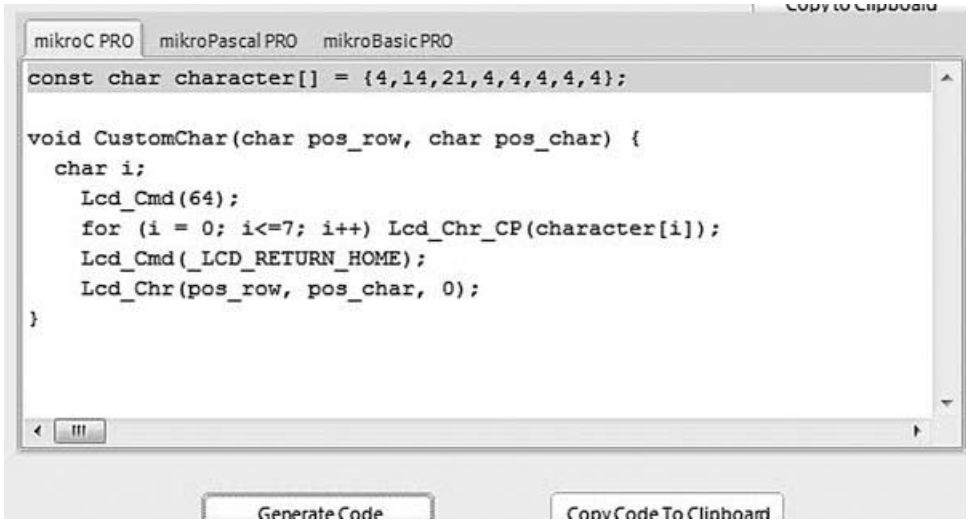


Figure 11.13 Generating code for the font

www.it-ebooks.info

```
BEGIN
        Define microcontroller – LCD connections
        Define bit map of the required font
        Configure PORT B as digital and output
        Initialise LCD
        Display text on LCD
        CALL CustomChar to display the created font
END


BEGIN/CustomChar
        Display required font as character 0
END/CustomChar
```

Figure 11.14 PDL of the project

11.5.1 Block Diagram

The block diagram of the project is shown in Figure 11.17.

11.5.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 11.18. The LCD is connected to PORT B, as in the earlier projects. The STRT button is connected to pin RC0. This pin is normally at logic 1 and goes to logic 0 when the button is pressed. The project is based on a PIC18F45K22 type microcontroller with an 8 MHz crystal clock, although most other PIC microcontrollers can also be used.

The connections between the microcontroller and the LCD are as follows:

LCD Pin	Microcontroller Pin
D4	RB0
D5	RB1
D6	RB2
D7	RB3
R/S	RB4
E	RB5

The contrast of the LCD is controlled by connecting a 10 KB potentiometer to pin 3 of the LCD. The microcontroller is Reset using an external push-button.

11.5.3 Project PDL

The PDL of this project is given in Figure 11.19.

11.5.4 Project Program

The program is named LCD5.C and the program listing of the project is shown in Figure 11.20. At the beginning of the project, the connections between the microcontroller and the LCD are

www.it-ebooks.info

```
/*************************************************************************************************
                        CREATING CUSTOM FONT ON LCD
                        ===========================

This project displays a custom font on the LCD. An "up arrow" is displayed with text as
shown below:

                Up arrow - <up arrow symbol>

The font has been created using the mikro C font editor.

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC  microcontroller can be
used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

  LCD pin  Microcontroller pin
   D4          RB0
   D5          B1
   D6          RB2
   D7          RB3
   R/S         RB4
   E           RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD. Other
pins of the potentiometer are connected to power and ground.


Author:  Dogan Ibrahim
Date:    November, 2011
File:    LCD4.C
*************************************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
```

Figure 11.15 Program listing of the project

www.it-ebooks.info

```
// The following code is generated automatically by the mikroC compiler font editor
//
const char character[] = {4,14,21,4,4,4,4,4};

void CustomChar(char pos_row, char pos_char) {
 char i;
   Lcd_Cmd(64);
   for (i = 0; i<=7; i++) Lcd_Chr_CP(character[i]);
   Lcd_Cmd(_LCD_RETURN_HOME);
   Lcd_Chr(pos_row, pos_char, 0);
}


//
// Start of Main program
//
void main()
{

   ANSELB = 0;                              // Configure PORT B as digital
   TRISB = 0;                               // Configure PORT B pins as output

   Lcd_Init();                              // Initialize LCD
   Lcd_Cmd(_LCD_CLEAR);                     // Clear display
   Lcd_Cmd(_LCD_CURSOR_OFF);                // Cursor off

   Lcd_Out(1, 1, "Up arrow -");             // Display text "Up arrow -"
   CustomChar(1, 12);                       // Display the "up arrow" symbol

   while(1);                                // End of program, wait here forever
}
```

Figure 11.15 (*Continued*)

Figure 11.16 The LCD display

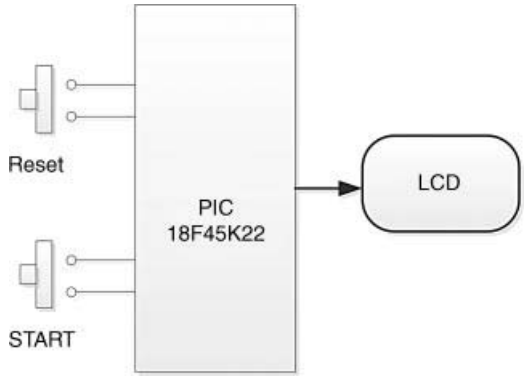


Figure 11.17 Block diagram of the project

www.it-ebooks.info

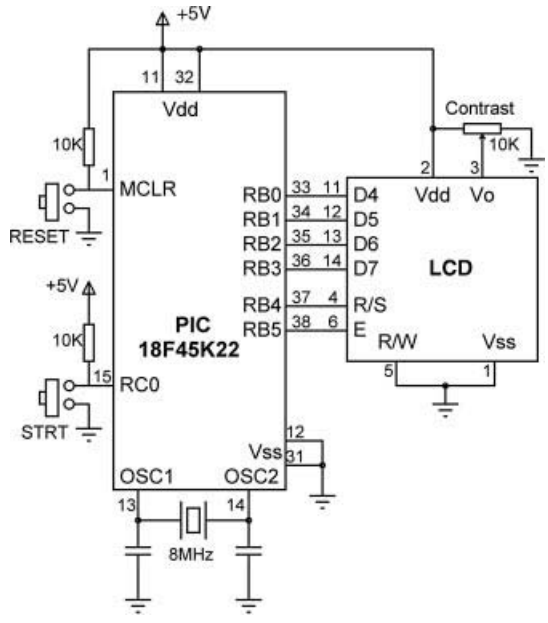


Figure 11.18 Circuit diagram of the project

BEGIN
Define STRT connection
Define the connection between LCD and microcontroller
Configure PORT B and PORT C as digital
Configure PORT B as output
Configure RC0 as input
Initialise LCD
Turn OFF cursor
Clear LCD
DO FOREVER
Wait Until STRT is pressed
CALL Number to get first dice number
CALL Number to get second dice number
Convert dice numbers to characters
Display text "Good Luck" on first row
Display the two dice numbers on second row
Wait 5 seconds
Clear LCD
ENDDO
END

BEGIN/Number
Generate a random number between 1 and 6
Return the generated number to the calling program
END/Number

Figure 11.19 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************************
                        LCD DICE
                        =======

This project displays two dice numbers on the LCD. The numbers are generated when
button STRT (connected to port pin RC0) is pressed. The display is ON for 5 seconds
and after this time the display is cleared, ready to generate two new numbers when
the STRT button is pressed.

The text "Good Luck" is displayed in row 1, and the numbers are displayed in
the second row in the following format:

    1234567890123456
      Good Luck
       X Y

where, X and Y are the two dice numbers.

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC microcontroller
can be used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

 LCD pin        Microcontroller pin
   D4             RB0
   D5             RB1
   D6             RB2
   D7             RB3
   R/S            RB4
   E              RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD
LCD is controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD
Other pins of the potentiometer are connected to power and ground.

Author:  Dogan Ibrahim
Date:    November, 2011
File:    LCD5.C
*************************************************************************************************/
unsigned char seed = 1;

sbit STRT at RC0_bit;

// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
```

Figure 11.20 Program listing of the project

www.it-ebooks.info

```
sbitLCD_RS_Direction at TRISB4_bit;
sbitLCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
// Pseudo random number generator function
//
unsigned char Number(int Lim, int Y)
{
unsigned char Result;
static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
return Result;
}

//
// Start of main program
//
void main()
{
unsigned char Num1, Num2;

ANSELB = 0;                              // Configure PORT B as digital
ANSELC = 0;                              // Configure PORT C as digital
TRISB = 0;                               // PORT B is output
TRISC = 1;                               // RC0 is input

Lcd_Init();                              // Initialise LCD
Lcd_Cmd(_LCD_CURSOR_OFF);                // Disable Cursor
Lcd_Cmd(_LCD_CLEAR);                     // Clear Display

for(;;)
  {
while(STRT == 1);                        // Wait until STRT is pressed

Num1 = Number(6, seed);                  // Generate first Number between 1 and 6
Num2 = Number(6, seed);                  // Generate second Number
Num1 = Num1 + '0';                       // Convert number to character
Num2 = Num2 + '0';                       // Convert number to character
Lcd_Out(1, 5, "Good Luck");              // Display "Good Luck"
Lcd_Chr(2,7, Num1);                      // Display first number
Lcd_Chr_Cp(' ');                         // Leave a space
Lcd_Chr_Cp(Num2);                        // Display second character
Delay_Ms(5000);                          // Wait 5 seconds
Lcd_Cmd(_LCD_CLEAR);                     // Clear display
  }
}
```

Figure 11.20 (*Continued*)

www.it-ebooks.info

defined using *sbit* statements. Port pin RC0 is given the name START, PORT B is configured as a digital output port, the LCD is initialised, cleared, and the cursor is turned OFF.

The program executes in an endless loop formed using a *for* statement. Inside this loop, the program waits until the START button is pressed. When the button is pressed, the random number generating function *Number* is called twice to generate two dice numbers between 1 and 6. This function receives two arguments: the range of numbers to be generated, and the seed value. The generated numbers are then converted into characters so that they can be displayed on the LCD. Text 'Good Luck' is displayed on the first row, starting from column 5. Then the first number is displayed on the second row, starting from column 7, using function Lcd_Chr. The second number is displayed after a space character using function Lcd_Chr_Cp. The numbers are displayed for 5 seconds, and after this time the display is cleared and the program is ready to generate two new dice numbers when the START button is pressed.

11.6 PROJECT 11.6 – Digital Voltmeter

This project is about the design of a digital voltmeter device with LCD output. The device can be used to measure voltages from 0 V to +5000 mV.

The voltage is measured and displayed every second. The measured voltage will be displayed in millivolts and in floating point format as follows:

```
V = nn.nnnnnn
```

11.6.1 Block Diagram

The block diagram of the project is shown in Figure 11.21.

11.6.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 11.22. The LCD is connected to PORT B as before. The analogue voltage to be measured is applied to port pin RA0 of the

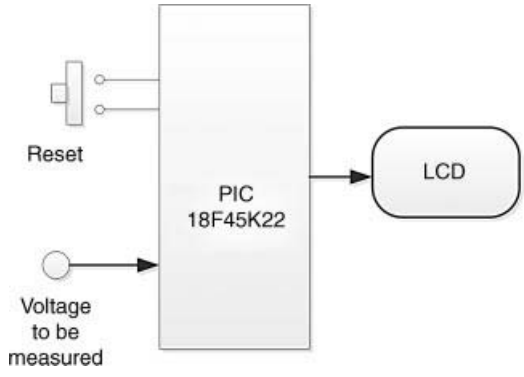


Figure 11.21 Block diagram of the project

www.it-ebooks.info

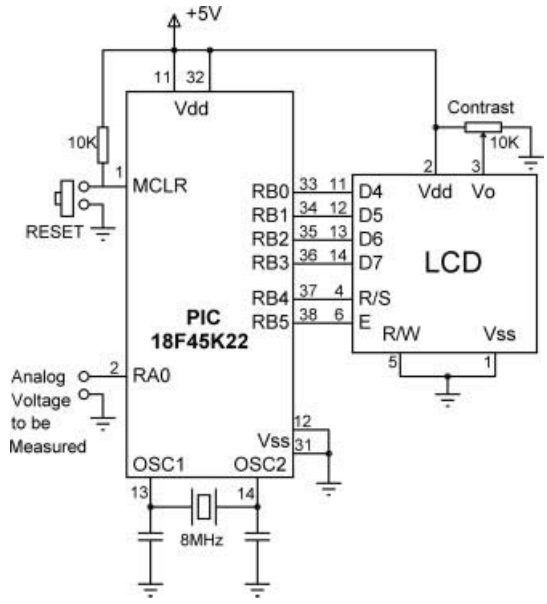


Figure 11.22 Circuit diagram of the project

microcontroller, and this pin is configured as an analogue input pin. The project is based on a PIC18F45K22 type microcontroller with an 8 MHz crystal clock, although most other PIC microcontrollers supporting A/D converters can also be used.

The connections between the microcontroller and the LCD are as follows:

LCD Pin	Microcontroller Pin
D4	RB0
D5	RB1
D6	RB2
D7	RB3
R/S	RB4
E	RB5

The contrast of the LCD is controlled by connecting a 10 KB potentiometer to pin 3 of the LCD. The microcontroller is Reset using an external push-button.

If you are using the EasyPIC 7 development board, then connect jumper J15 across pins RA0, and then you can vary the voltage at this pin for testing, by rotating the arm of the potentiometer located next to the port pins. Also make sure that the jumpers Read-X and Read-Y of SW3 are in the OFF positions.

11.6.3 Project PDL

The PDL of this project is very simple and is given in Figure 11.23.

www.it-ebooks.info

```
BEGIN
    Define connection between the microcontroller and LCD
    Configure PORT A as analog and PORT B as digital
    Configure pin RA0 as input
    Configure PORT B as digital
    Initialise LCD
    Initialise A/D converter
    DO FOREVER
        Read analog voltage from A/D converter Channel 0
        Convert voltage read to millivolts
        Display voltage on the LCD
        Wait 1 second
        Clear LCD
    END
END
```

Figure 11.23 PDL of the project

11.6.4 Project Program

The program is named LCD6.C and the program listing of the project is shown in Figure 11.24. At the beginning of the project, the connection between the microcontroller and the LCD are defined using *sbit* statements. PORT A is configured as an analogue port and pin RA0 is configured as an input pin. Then, PORT B is configured as a digital output port, the LCD is initialised, cleared, and the cursor is turned OFF. The A/D converter module is initialised by calling function ADC_Init. This function sets the A/D converter to default configuration, where the clock is derived from the internal RC circuit, and the reference voltage is set to +5 V.

The program is executed in an endless loop formed using a *for* statement. Inside this loop, the input voltage is converted into digital using the A/D converter of the microcontroller. Function ADC_Get_Sample(0) reads the analogue voltage from Channel 0 (i.e. port pin RA0) of the microcontroller. Here, the converted digital data is stored in variable *Converted-Data*. The A/D converters on PIC18F45K22 microcontroller are 10-bits wide, providing 1024 steps. Thus, for a reference voltage of +5 V, each step corresponds to 4.88 mV, and this is the minimum voltage change that can be detected by our voltmeter. The digital data is then converted into millivolts and stored in variable *mV* after multiplying with 5000.0 and dividing by 1024.0. This data is converted into a string using function FloatToStr, so that it can be displayed on the LCD. The text 'V = ' is displayed first starting from row 1, column 1 of the LCD. Then, the measured voltage is displayed in floating point format. The program then waits for 1 second, clears the display, and the above process is repeated.

11.7 PROJECT 11.7 – Temperature and Pressure Display

This project is about the design of a microcontroller based device to measure the ambient temperature and the pressure and to display them on a 2 × 16 LCD. The temperature will be displayed on the first row and the pressure on the second row. The display is in floating point format as follows:

```
T(C) = nnn.nnnn
P(mb) = nnn.nnnn
```

www.it-ebooks.info

```
/*************************************************************************************************
                                    Digital Voltmeter
                                    ==========

This project measures analog voltages in the range 0 to +5V and displays on the LCD.
The measurement is done every second.

The measured voltage is displayed in millivolts and in floating point format for higher
accuracy. The format of the display is as follows:

        V = nn.nnnnn

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type
microcontroller, operated from an 8MHz crystal (any other type PIC microcontroller with
A/D converter can be used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

 LCD pin     Microcontroller pin
   D4          RB0
   D5          RB1
   D6          RB2
   D7          RB3
   R/S         RB4
   E           RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD. Other
pins of the potentiometer are connected to power and ground.

The analog voltage to be measured is connected to port pin RA0 of the microcontroller.
The A/D converter on the PIC18F45K22 microcontroller is 10-bits wide and thus with a
reference voltage of +5V, the minimum voltage that can be detected is 4.88mV.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    LCD6.C
*************************************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
```

Figure 11.24 Program listing of the project

There are many analogue and digital temperature sensors available. The one used in this project is the LM35DZ integrated circuit analogue temperature sensor. This is a small 3-pin sensor, where one of the pins is connected to +V, the other one to GND, and the third one is the output pin. The output voltage is proportional to temperature and is given by

$$\text{Vo} = 10\,\text{mV}/^{\circ}\text{C} \tag{11.1}$$

www.it-ebooks.info

```
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
// Start of main program
//
void main()
{
  unsigned int ConvertedData;
  float mV;
  unsigned char Head[] = "V = ";
  unsigned char Txt[15];

  ANSELA = 1;                          // Configure PORT A as analog
  ANSELB = 0;                          // Configure PORT B as digital
  TRISB = 0;                           // PORT B is output
  TRISA = 1;                           // RA0 is input

  Lcd_Init();                          // Initialise LCD
  Lcd_Cmd(_LCD_CURSOR_OFF);            // Disable Cursor
  Lcd_Cmd(_LCD_CLEAR);                 // Clear Display

  ADC_Init();                          // Initialise the A/D converter module

  for(;;)                              // DO FOREVER
  {
   ConvertedData = ADC_Get_Sample(0);  // Read analog data from Channel 0
   mV = ConvertedData * 5000.0 / 1024.0; // Convert to millivolts as floating point
   FloatToStr(mV, Txt);                // Convert to character in array Txt

   Lcd_Out(1, 1, Head);                // Display "V = "
   Lcd_Out_Cp(Txt);                    // Display the voltage in mV
   Delay_Ms(1000);                     // Wait 1 second
   Lcd_Cmd(_LCD_CLEAR);                // Clear display
  }
}
```

Figure 11.24 (*Continued*)

Thus, for example, at 10°C the output voltage is 100 mV, at 25°C the output voltage is 250 mV, and so on.

The ambient pressure sensor used in this project is the MPX4115A. This sensor generates an analogue voltage proportional to the ambient pressure. The device is available either in a 6-pin or an 8-pin package.

The pin configuration of a 6-pin sensor is:

Pin	Description
1	Output voltage
2	Ground
3	+5 V supply
4–6	not used

www.it-ebooks.info

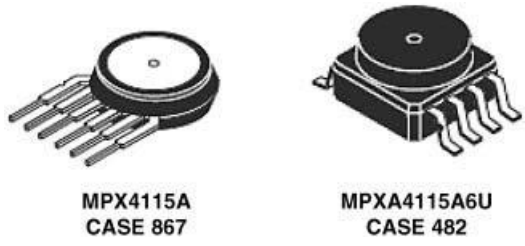


Figure 11.25 MPX4115A pressure sensors

and for an 8-pin sensor:

Pin	Description
1	not used
2	+5 V supply
3	Ground
4	Output voltage
5–8	not used

Figure 11.25 shows pictures of this sensor with both types of pin configurations.
The output voltage of the sensor is given by

$$V = 5.0^{*}(0.009^{*}kPa - 0.095) \tag{11.2}$$

or

$$kPa = \frac{\frac{V}{5.0} + 0.095}{0.009} \tag{11.3}$$

where

kPa = atmospheric pressure (Kilo Pascals);
V = output voltage of the sensor (V).

The atmospheric pressure measurements are usually shown in millibars. At sea level and at a temperature of 15°C, the atmospheric pressure is 1013.3 millibars. In equation 11.3 the pressure is given in kPa. To convert kPa to millibars, we have to multiply equation 11.3 by 10 to give:

$$mb = 10x\frac{\frac{V}{5.0} + 0.095}{0.009} \tag{11.4}$$

or

$$mb = \frac{2.0\,V + 0.95}{0.009} \tag{11.5}$$

www.it-ebooks.info

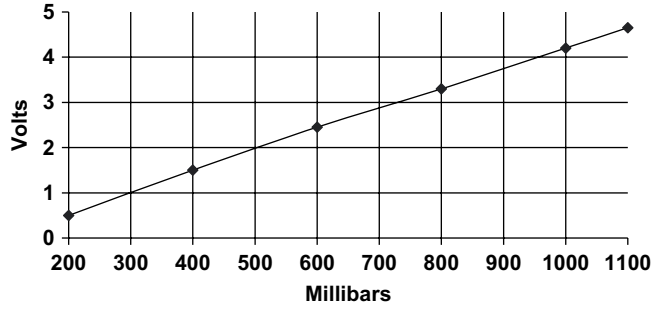


Figure 11.26 Variation of sensor output voltage with pressure

or

$$mb = \frac{2000\,mV + 950}{9} \tag{11.6}$$

where the voltage is in millivolts.

Figure 11.26 shows the variation of the output voltage of the MPX4115A sensor as the pressure is varied. Our range of interest lies in the area where the ambient (atmospheric) pressure is between 800 and 1100 millibars.

11.7.1 Block Diagram

The block diagram of the project is shown in Figure 11.27. The top row of the LCD shows the temperature and the bottom row the pressure.

11.7.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 11.28. The LCD is connected to PORT B, as in the earlier projects. The temperature sensor is connected to analogue input RA0

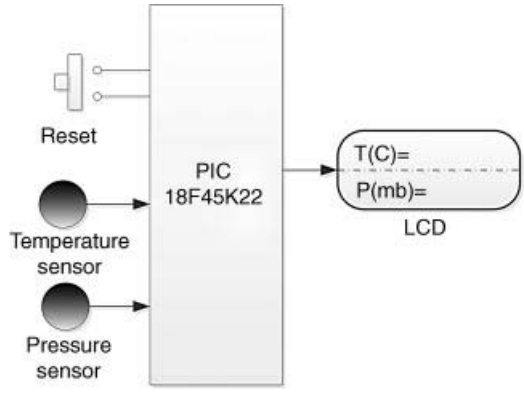


Figure 11.27 Block diagram of the project

www.it-ebooks.info

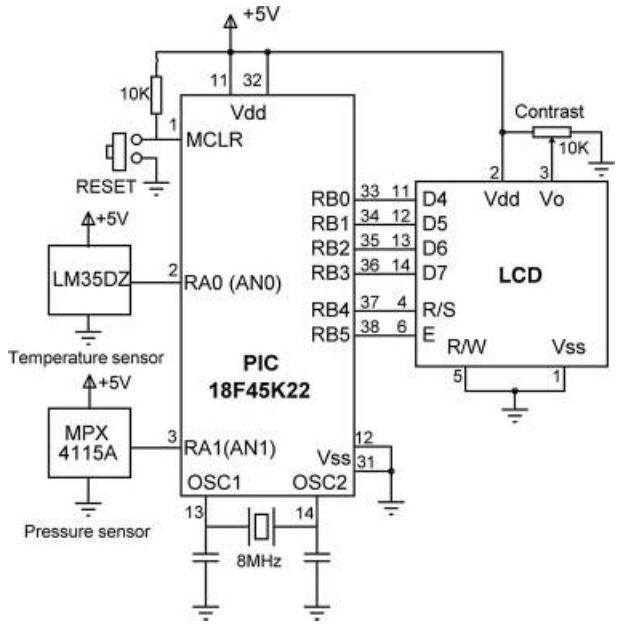


Figure 11.28 Circuit diagram of the project

(Channel 0) of the microcontroller. In a similar way, the pressure sensor is connected to the analogue input RA1 (Channel 1) of the microcontroller. Both RA0 and RA1 are configured as analogue input ports. The project is based on a PIC18F45K22 type microcontroller with an 8 MHz crystal clock, although most other PIC microcontrollers supporting at least two A/D converters can also be used.

If you are using the EasyPIC 7 development board, then make sure that jumper J15 is not connected across RA0 or RA1 inputs.Also make sure that the jumpers Read-X and Read-Y of SW3 are in the OFF positions.

11.7.3 Project PDL

The PDL of this project is given in Figure 11.29.

11.7.4 Project Program

The program is named LCD7.C and the program listing of the project is shown in Figure 11.30. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements. PORT A is configured as an analogue port and pins RA0 and RA1 are configured as input pins. Then, PORT B is configured as a digital output port, the LCD is initialised, cleared, and the cursor is turned OFF. The A/D converter module is initialised by calling function ADC_Init. This function sets the A/D converter to default configuration where the clock is derived from the internal RC circuit, and the reference voltage is set to +5 V.

The program is executed in an endless loop formed using the *for* statement. Inside this loop, the output of LM35DZ is read using the A/D converter of the microcontroller. Function

www.it-ebooks.info

```
BEGIN
    Define connections between the LCD and microcontroller
    Configure PORT A pins RA0 and RA1 as analog inputs
    Configure PORT B as digital output
    Initialise LCD
    Turn OFF cursor
    Clear display
    Initialise A/D converter
    DO FOREVER
        Read temperature from Channel 0
        Convert to millivolts
        Convert to Degrees Centigrade
        Convert to a string
        Read pressure from Channel 1
        Convert to millivolts
        Convert to millibars
        Convert to a string
        Display temperature on top row of LCD
        Display pressure at bottom row of LCD
        Wait 1 second
        Clear display
    ENDDO
END
```

Figure 11.29 PDL of the project

ADC_Get_Sample(0) reads the analogue voltage from Channel 0 (i.e. port pin RA0) of the microcontroller. Here, the converted digital data is stored in variable *TemperatureData*. The A/D converters on PIC18F45K22 microcontroller are 10 bits wide, providing 1024 steps. Thus, for a reference voltage of +5 V, each step corresponds to 4.88 mV, and this is the minimum voltage change that can be detected by our voltmeter. The digital data is then converted into millivolts and divided by 10 to find the temperature in °C. The temperature is stored in variable *Temperature* as a floating point number.

Then, the pressure is read from Channel 1 (i.e. port pin RA1) and stored in variable *PressureData* as a digital value. This value is then converted into volts and the pressure is calculated using equation 11.6 and stored in variable *Pressure* as a floating point number.

In the final part of the program, both the temperature and pressure are converted into strings using the built-in function FloatToStr and the resulting strings are displayed on the top and bottom rows of the LCD. Notice that the temperature reading is displayed as 'T(C) = nnn.nnnn', and the pressure reading is displayed as 'P(mb) = nnn.nnnn'. The program waits for 1 second and the above process is repeated forever.

Figure 11.31 shows a typical display of the temperature and pressure.

11.8 PROJECT 11.8 – The High/Low Game

This project uses a 4×4 keypad to create the classical a High/Low game. For those of you who are not familiar with the game, here are the rules for this version of the game:

- The computer will generate a secret random number between 1 and 32 767.
- The top row of the LCD will display 'Guess Now . . . '

www.it-ebooks.info

```
/*************************************************************************************************************
                        Temperature and Pressure Display
                        ================================

This project measures the ambient temperature and pressure and displays the values on an LCD.
The temperature is displayed in Degrees Centigrades on the top row, and the pressure is
displayed in millibars on the bottom row in the following format:

  T(C)=
  P(mb)=

Both dislays are in floating point format for higher accuracy.

The temperature is sensed using a LM35DZ type analog temperature sensor. This sensor gives an
output voltage proportional to the measured temperature.

The pressure is sensed using a MPX4115A type analog pressure sensor.

The measurements are done every second.

The measured values is displayed in millivolts and in floating point format for higher accuracy.

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type

microcontroller, operated from an 8MHz crystal (any other type PIC microcontroller with at

least two A/D converters can be used if desired).

The LCD is connected to PORT B of the microcontroller as follows:

 LCD pin      Microcontroller pin
   D4           RB0
   D5           RB1
   D6           RB2
   D7           RB3
   R/S          RB4
   E            RB5

The temperature sensor is connected to analog port RA0 (AN0) and the pressure sensor is
connected to analog port RA1 (AN1).

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10K potentiometer to pin Vo of the LCD. Other pins
of the potentiometer are connected to power and ground.

Author: Dogan Ibrahim
Date:   December, 2011
File:   LCD7.C
*************************************************************************************************************/

// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
```

Figure 11.30 Program listing of the project

www.it-ebooks.info

```
sbitLCD_RS_Direction at TRISB4_bit;
sbitLCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
// Start of main program
//
void main()
{
  unsigned int TemperatureData, PressureData;
  float mV;
  unsigned char T[] = "T(C)=              ";
  unsigned char P[] = "P(mb)=              ";

  ANSELA = 3;                                   // Configure PORT A as analog
  ANSELB = 0;                                   // Configure PORT B as digital
  TRISB = 0;                                    // PORT B is output
  TRISA = 3;                                    // RA0 and RA1 are inputs

  Lcd_Init();                                   // Initialise LCD
  Lcd_Cmd(_LCD_CURSOR_OFF);                     // Disable Cursor
  Lcd_Cmd(_LCD_CLEAR);                          // Clear Display

  ADC_Init();                                   // Initialise the A/D converter module

  for(;;)                                       // DO FOREVER
  {
//
// Read and process the Temperature
//
  TemperatureData = ADC_Get_Sample(0);          // Read temperature data from Channel 0
  mV = TemperatureData * 5000.0 / 1024.0;       // Convert to millivolts as floating point
  mV = mv / 10.0;                               // Convert to Degrees Centigrade
  FloatToStr(mV, T+5);                          // Convert to character in array T
//
// Read and process the Pressure
//
  PressureData = ADC_Get_Sample(1);             // Read pressure data from Channel 1
  mV = PressureData * 5000.0 / 1024.0;          // Convert to millivolts as floating point
  mV = 2000.0*mv + 950.0;                       // Convert to millibars
  mV = mV / 9.0;
  FloatToStr(mV, P+6);                          // Convert to character in array P
//
// Now display the temperature and pressure
//
  Lcd_Out(1, 1, T);                             // Display temperature
  Lcd_Out(2, 1, P);                             // Display pressure
  Delay_Ms(1000);                               // Wait 1 second
  Lcd_Cmd(_LCD_CLEAR);                          // Clear display
 }
}
```

Figure 11.30 (*Continued*)

www.it-ebooks.info

Figure 11.31 Typical display of temperature and pressure

- The player will try to guess what the number is, by entering a number on the keypad and then pressing the ENTER key.
- If the guessed number is higher than the secret number, the bottom row of the LCD will display 'HIGH – Try Again'.
- If the guessed number is lower than the secret number, the bottom row of the LCD will display 'LOW – Try Again'.
- If the player guesses the number, then the bottom row will display 'Well Done . . . '
- The program waits for 5 seconds and the game re-starts automatically.

Before going into the design of the project, it is worthwhile to learn a bit more about how the keypads work and the mikroC Pro for PIC commands available to use a keypad.

11.8.1 Keypads

A 4 × 4 keypad consists of 16 keys with internal mechanical switches at each key position. Figure 11.32 shows a typical 4 × 4 keypad connected to PORT D of a PIC microcontroller.

The keypad operates on the principle of 'scan and detect', where in Figure 11.32, RD0, RD1, RD2 and RD3 are configured as inputs and connected to columns of the keypad via

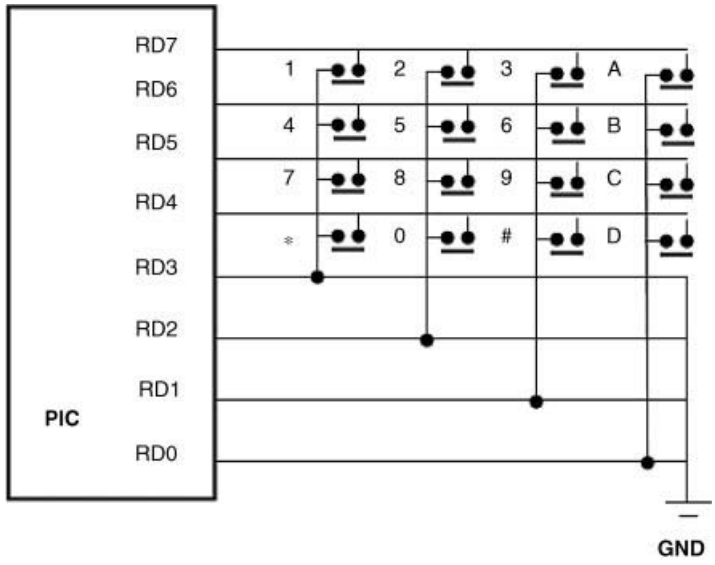


Figure 11.32 A 4 × 4 keypad connected to a microcontroller

www.it-ebooks.info

pull-down resistors. RD4, RD5, RD6 and RD7 pins are configured as outputs and connected to rows of the keypad. The program sends logic 1 to each row of the keypad in turn and checks the columns by reading the logic state at the column. Pressing any key will cause logic 1 to be applied to input pins of the microcontroller. The software detects which key is pressed by scanning the inputs. For example, assume that while sending out logic 1 to row 2 where numbers 4, 5, 6, B are, we detect that RD2 input is logic 1. In this case, the pressed key must be number 5, and so on.

11.8.2 mikroC Pro for PIC Keypad Library Functions

mikroC Pro for PIC provides a built-in library called 'Keypad Library', which contains functions to help us use keypads in our programs. These functions are given below:

- Keypad_Init
- Keypad_Key_Press
- Keypad_Key_Click

11.8.2.1 Keypad_Init

This function initialises the keypad library. The port where the keypad is connected must be declared before this function is called using the reserved name 'keypadPort'. For example, if the keypad is connected to PORT D, then at the beginning of the program we must declare:

```
charkeypadPort at PORTD;
```

11.8.2.2 Keypad_Key_Press

This function reads the key from the keypad when it is pressed. The code of the key is returned as a number between 1 and 16. If no key is pressed, a 0 is returned.

11.8.2.3 Keypad_Key_Click

This function is similar to Keypad_Key_Press, but this is a blocking function. The function waits until a key is pressed. If more than one key is pressed, the function will wait for all the keys to be released and then return code of the first key pressed.

11.8.3 Generating a Random Number

In our program, we will be generating a random integer number using the mikroC Pro for PIClibrary functions 'srand' and 'rand'. Function 'srand' must be called with an integer argument (or 'seed') to prepare the random number generator library. Then, every time function 'rand' is called, a new random number will be generated between 1 and 32 767. The set of numbers generated are the same if the program is re-started with the same 'seed' applied to function 'srand'. Thus, if the game is re-started after resetting the microcontroller, the same set of numbers will be generated.

www.it-ebooks.info

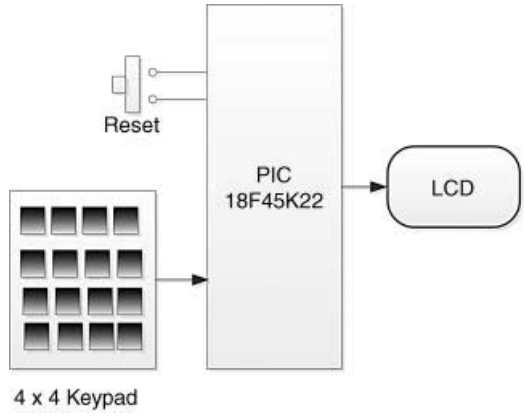


Figure 11.33 Block diagram of the project

11.8.3.1 Block Diagram

The block diagram of the project is shown in Figure 11.33. The keypad is organised, as shown in Figure 11.34.

The numbers returned by the mikroC Pro for PICkeypad library when a key is pressed is as follows:

Key pressed	Number returned
1	1
2	2
3	3
A	4
4	5
5	6
6	7
B	8
7	9
8	10
9	11
C	12
*	13
0	14
#	15
D	16

We will be using key 'D' as the ENTER key in our program. Also, we will be correcting the key numbering in our program so that, for example, when '7' is pressed on the keypad, a 7 is returned and not a 9 as in the above table.

Figure 11.35 shows a typical 4×4 keypad, manufactured by mikroElektronika (http://www.mikroe.com). This keypad can be directly plugged into the PORT D connector at the edge of the EasyPIC 7 development board.

www.it-ebooks.info

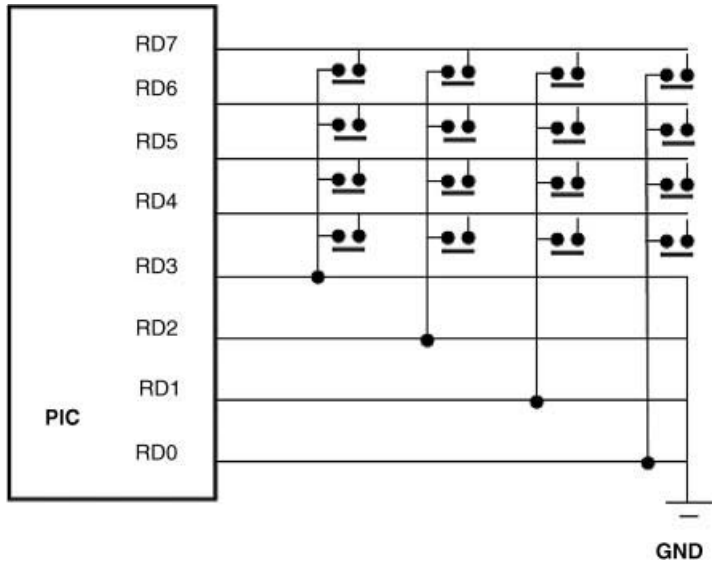


Figure 11.34 Key configuration on the keypad

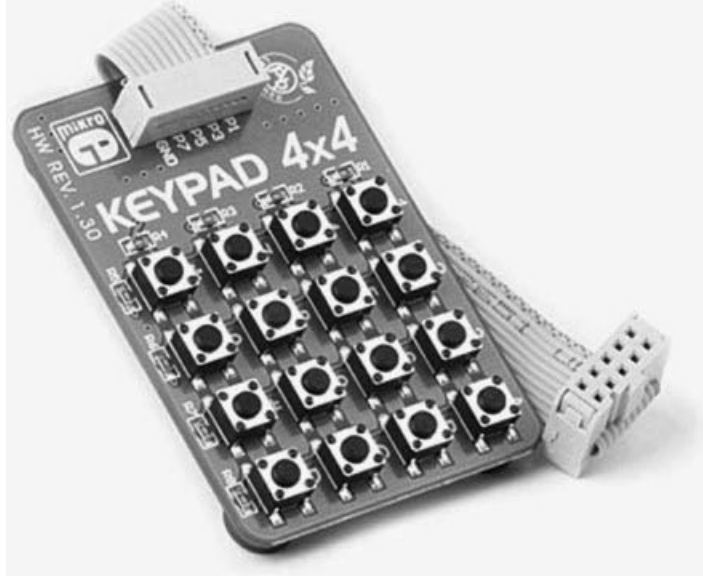


Figure 11.35 A typical 4 × 4 keypad

www.it-ebooks.info

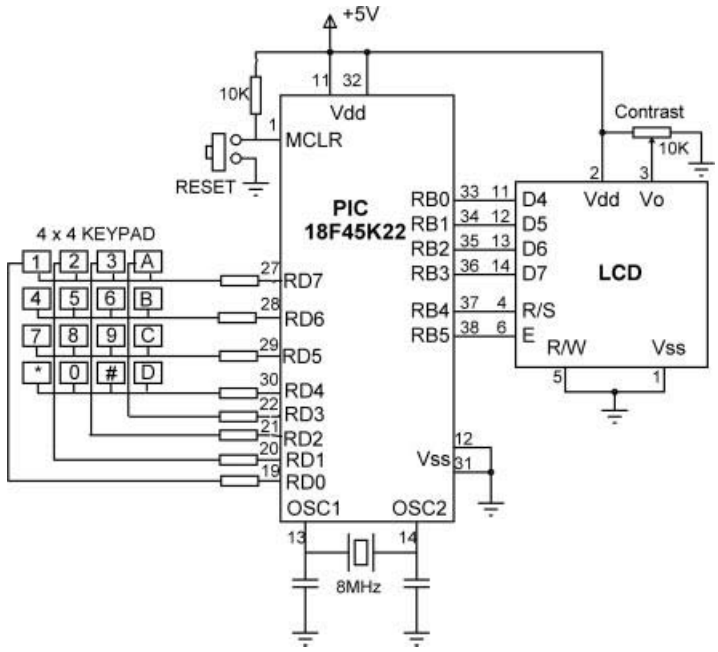


Figure 11.36 Circuit diagram of the project

11.8.3.2 Circuit Diagram

The circuit diagram of the project is shown in Figure 11.36. The LCD is connected to PORT B, as in the earlier projects. The rows and columns of the keypad are connected to upper and lower nibbles of PORT D, respectively.

11.8.3.3 Project PDL

The PDL of this project is given in Figure 11.37.

11.8.3.4 Project Program

The program is named LCD8.C and the program listing of the project is shown in Figure 11.38.

At the beginning of the program, keypadPORT is declared as PORTD and some other variables used in the program are also declared. Then PORTB is configured as digital I/O, keypad library is initialised, the LCD is initialised and message 'High/Low Game' is displayed on the LCD. After a 2 second delay, the program continues in an endless loop.

If this is a new game, the LCD is cleared and message 'Guess Now . . . ' is displayed on the first row of the LCD. Then a random number is generated between 1 and 32 767 by calling library function 'rand' and this number is stored in variable 'GuessNumber'. Notice that the 'srand' library function must be called with an integer number before calling 'rand'.

The keypad is then checked and numbers are received until the ENTER key (key D) is pressed. The key numbers are then adjusted such that if, for example, 4 is pressed, number 4

www.it-ebooks.info

```
BEGIN
    Declare keypad port number
    Define LCD to microcontroller pin connections
    Configure PORT B as digital
    Initialize keypad library
    Initialize LCD
    Display heading "High/Low Game"
    Set new game flag
    Wait 2 seconds
    DO FOREVER
        IF new game flag is set THEN
            Clear LCD
            Turn OFF cursor
            Generate a random number (secret number)
            Display "Guess Now.." on row 1
        ENDIF
        Read and display (on row 2) numbers until ENTER is pressed
        IF entered number > secret number THEN
            Display "HIGH –Try Again"
            Wait 1 second
            Clear second row of LCD
        ELSE IF entered number < secret number THEN
            Display "LOW –Try Again"
            Wait 1 second
            Clear second row of LCD
        ELSE IF entered number = secret number THEN
            Display "Well Done.."
            Wait 5 seconds
            Set new game flag
        ENDIF
    END
```

Figure 11.37 PDL of the project

is used by the program instead of 5. Similarly, if key 0 is pressed, number 0 is used by the program instead of 14 returned by the keypad library routine. The numbers entered by the player are displayed on the second row of the LCD, as they are entered so that the player can see what he/she has entered. After the player presses the ENTER key, a 1 second delay is introduced. The number entered by the player is stored in variable 'PlayerNumber' in decimal format.

The program then calculates the difference between the secret number in 'GuessNumber' and the number entered by the player (in PlayerNumber). This difference is stored in variable 'Diff'.

If 'Diff' is positive, that is if the number entered by the player is greater than the secret number, then the program displays message 'HIGH - Try Again', waits for 1 second, and clears the second row of the LCD, ready for the player to try another number.

If 'Diff' is negative, that is if the number entered by the player is less than the secret number, then the program displays message 'LOW - Try Again', waits for 1 second, and clears the second row of the LCD, ready for the player to try another number.

If 'Diff' is 0, that is if the number entered by the player is equal to the secret number, then the program displays message 'Well Done . . . ' waits for 5 seconds, and sets the

www.it-ebooks.info

```
/*************************************************************************************************
                        High/Low Game Using Keypad
                        ==========================

This project implements the High/Low game using the 4 x 4 kaypad and the LCD. The
program generates a random number between 1 and 32767 and expects the player to
guess the number. The LCD displays "Guess Now.." on top row of the display.

The player then guesses the number by entering a number via the keypad and then
pressing the ENTER key. If the guessed number is bigger than the generated number
the message "HIGH - Try Again" will be displayed on bottom row of the LCD.

If the guessed number is lower than the generated number then the message
"LOW -Try Again" will be generated on the bottom row of the LCD. If on the other hand
the player guesses the number correctly, the bottom row of the LCD will display the
message "Well Done..".

The game will re-start after a delay of 5 seconds.

The microcontroller in this project is PIC18F45K22 and is operated from an 8MHz
crystal as before.

Programmer:   Dogan Ibrahim
File:         LCD8.C
Date:         December, 2011

*************************************************************************************************/
char keypadPORT at PORTD;
unsigned char kp, new_game;
unsigned int GuessNumber, PlayerNumber;
int Diff;

// Declare LCD connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD connections


void main()
{
```

Figure 11.38 Program listing of the project

www.it-ebooks.info

```
unsigned char Txt1[4];

ANSELB = 0;                             // Configure PORT B as digital
Keypad_Init();                          // Initialize keypad library
Lcd_Init();                             // Initialize LCD
Lcd_Out(1, 1, "High/Low Game");         // Display heading
Delay_ms(2000);                         // Wait 2 seconds

new_game = 1;
srand(5);                               // Generate a random number sequence

for(;;)                                 // DO FOREVER
{
        if(new_game == 1)
        {
                Lcd_Cmd(_LCD_CLEAR);            // Clear LCD
                Lcd_Cmd(_LCD_CURSOR_OFF);       // Turn OFF cursor
                Lcd_Out(1, 1, "Guess Now..");   // Display "Guess Now.."
                GuessNumber = rand();           // Generate a random number
         }
        kp = 0;
        PlayerNumber = 0;
        Lcd_Out(2, 1, "");                              // Position cursor at 1,1
        while(kp != 16)                                 // Until ENTER pressed
        {
          do
          {
               kp = Keypad_Key_Click();                 // Look for key press
          }while(!kp);

          if(kp != 16)                                  // If not ENTER key
          {
                if(kp > 4 && kp <9)kp = kp-1;           // 5 is 4, 6 is 5….
                if(kp > 8 && kp < 12)kp = kp-2;         // 7 is 9, 8 is 10…
                if(kp == 14)kp=0;                       // 0 is14
                PlayerNumber = 10*PlayerNumber + kp;
                ByteToStr(kp, Txt1);
                Txt1[0] = Txt1[2];                      // Get the number
                Txt1[1] = '\0';                         // Make a string
                Lcd_Out_Cp(Txt1);                       // Display on LCD
          }
        }

       Delay_ms(1000);                                  // Wait one second
       Diff = PlayerNumber -GuessNumber;                // Find the diff

       if(Diff > 0)                                     // Greater ?
       {
                Lcd_Out(2, 1, "HIGH -Try Again");
                new_game = 0;                           // Not a new game
                Delay_ms(1000);
```

Figure 11.38 (*Continued*)

www.it-ebooks.info

Figure 11.39 Display from the game – start of the game

Figure 11.40 Display from the game – user guessed 258

Figure 11.41 Display from the game – the guess was low

'new_game' flag so that a new secret number can be generated by the program. The game continues as before.

Figures 11.39 to 11.41 show various displays from the game. Notice that the keypad keys are not debounced in the keypad library and sometimes you may get double key strokes, even though you press a key once. You should be firm and quick when pressing a key to avoid this to happen.

11.9 Summary

This chapter has described the use of text based LCD displays in microcontroller based projects. All the projects given in the chapter have been tested and are working. Early projects are about displaying text and numbers on LCDs. Then, more advanced projects are given, such as creating custom fonts, LCD dice project, LCD voltmeter project, and measuring the ambient temperature and pressure and displaying on the LCD. The last project is the classical High/Low game, where the microcontroller guesses a number and the user attempts to find this number. The microcontroller hints when the entered number is higher or lower than the number guessed.

www.it-ebooks.info

Exercises

11.1 It is required to design an event counter system. An event is recognised by the LOW to HIGH transition of an external digital signal. Assuming this signal is connected to port pin RC0 of a PIC microcontroller, write a program to display the number of events occurring.

11.2 Design a microcontroller based digital thermometer using the LM35DZ temperature sensor and display the temperature on the LCD. Use +5 V reference for the A/D converter.

11.3 Repeat exercise (2), but use a +3 V reference voltage for the A/D converter.

11.4 Repeat exercise (2), but display the temperature both in °C and in °F.

11.5 Repeat exercise (2), but use an external push-button switch such that the temperature is normally displayed in °C, but when the switch is pressed the temperature is displayed in °F.

www.it-ebooks.info

12

Graphics LCD Projects

In this chapter we will look at the design of projects using graphics LCDs (GLCDs). The GLCDs used in the projects in this section are 128 × 64 pixel monochrome, 78 × 70 × 14.3 mm displays, based on the KS108/KS107 controller. The projects are organised by increasing difficulty and the reader is recommended to follow the projects in the given order.

12.1 PROJECT 12.1 – Creating and Displaying a Bitmap Image

12.1.1 Project Description

This project shows how a monochrome bitmap image can be created and then displayed on the GLCD.

If a bitmap image is already available, we can use the mikroC Pro for PIC GLCD bitmap editor tool to convert the image into a data array, so that it could be used in a C program to display the image on the GLCD. Alternatively, we could create our own bitmap images using suitable programs. There are many tools for creating bitmap images. Perhaps the easiest way to create a bitmap image is by using the Windows Paint program, which is distributed free of charge with the Windows Operating System. In this project we will use the Paint program to create the image of a face with text and then display it on the GLCD. Notice that the created image or the available bitmap image must be monochromatic.

The steps in creating a bitmap image with the Paint program and then displaying on the GLCD are given below:

- Start the Windows Paint program. Start -> All Programs -> Accessories -> Paint.
- Click ‘Resize’. Unclick ‘Maintain Aspect Ratio’. Click ‘Pixels’. Select the GLCD screen size in pixels as Horizontal: 128, Vertical: 64. Click ‘OK’ to accept the entries (see Figure 12.1).
- Click the ‘Magnifier’ button and then click on the image to enlarge the image to a suitable size.
- Click ‘View’ and then select ‘Gridlines’. As shown in Figure 12.2, you should now see the GLCD screen with 128 pixels in the horizontal direction, and 64 pixels in the vertical direction.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

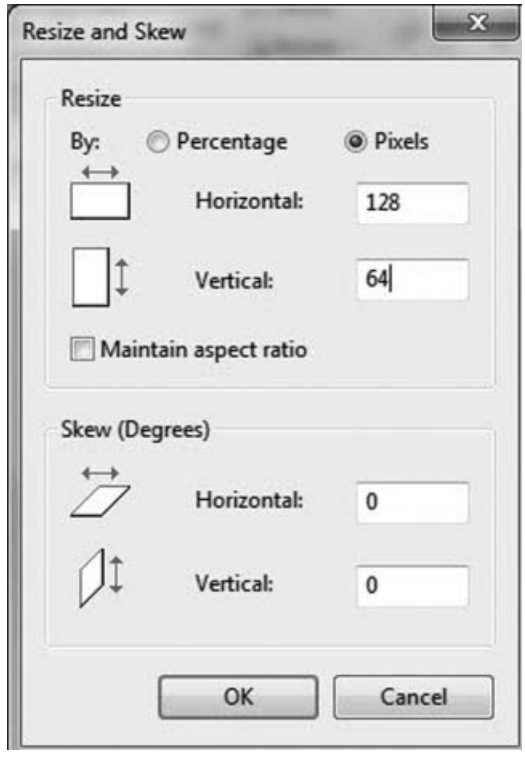


Figure 12.1 Select the GLCD screen size as 128 × 64 pixels

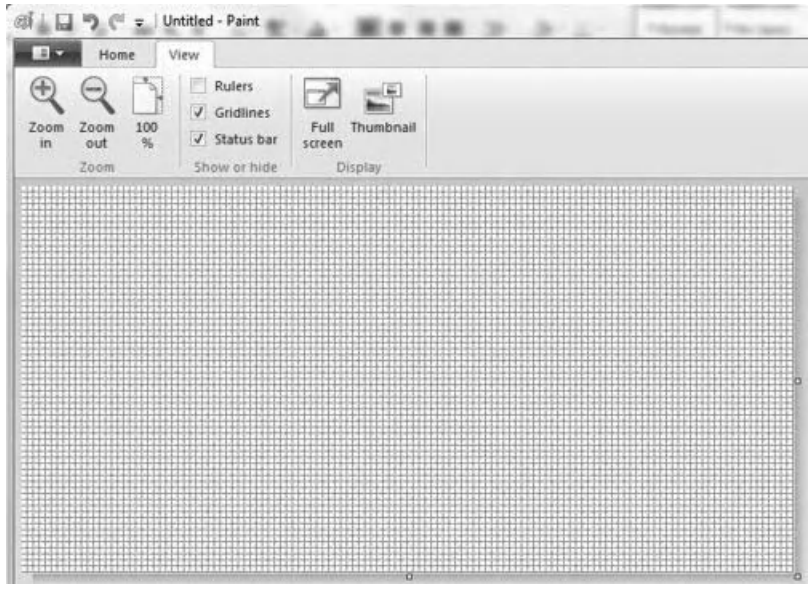


Figure 12.2 The GLCD screen

www.it-ebooks.info

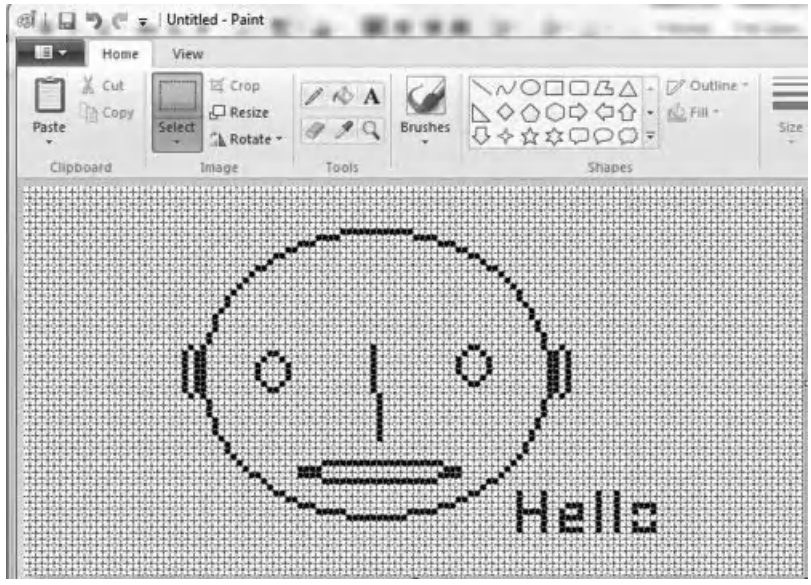


Figure 12.3 Created bitmap image

- We can now create our bitmap image. Click the 'Home' button so that we can select and use the various drawing tools. In this project, the simple bitmap image shown in Figure 12.3 is created as an example.
- Save the image as a bitmap (.BMP) file.
- Start mikroC Pro for PIC program. Select Tools -> GLCD Bitmap Editor.
- Select 'KS0108' controller.
- Click 'Load BMP' and enter the filename of the image created.
- As shown in Figure 12.4, you should see the image on the virtual GLCD. In addition, the bitmap image code will be shown at the bottom of the tool. The generated code is nothing more than a constant character array that contains information on which pixels of the GLCD should be illuminated to display the image. For a GLCD of 128×64 pixels, the size of the array would be 1024 bytes. Part of the generated code is shown below for illustration purposes:

```
//------------------------------
// GLCD Picture name: Untitled.bmp
// GLCD Model: KS0108 128x64
//------------------------------

const code char Untitled[1024] = {
0,   0,   0,   0,   0,  ..................................................
..........................................................................
..........................................................................
};
```

www.it-ebooks.info

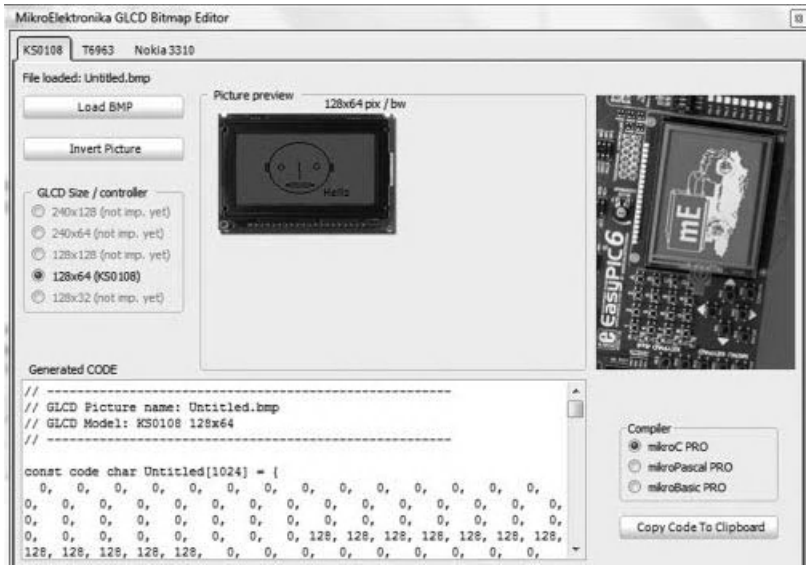


Figure 12.4 Using the GLCD Bitmap Editor

- Click 'Copy Code to Clipboard' to save the code in the clipboard for future use.
- We will see later in the project how to use the generated code to display the image on the GLCD.

12.1.2 Block Diagram

The block diagram of the project is shown in Figure 12.5.

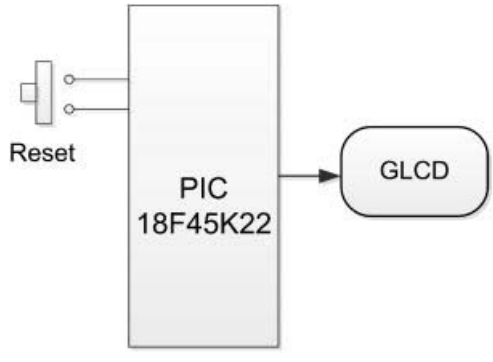


Figure 12.5 Block diagram of the project

www.it-ebooks.info

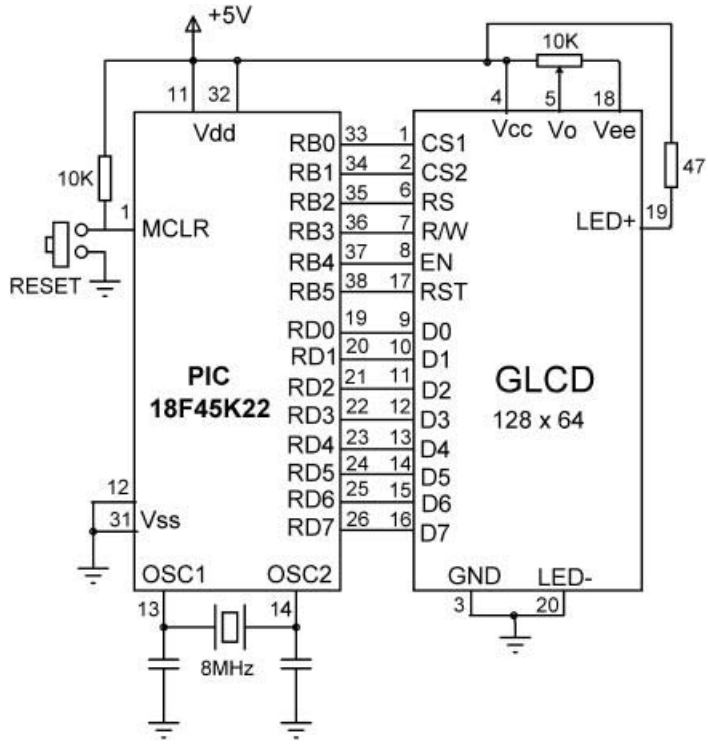


Figure 12.6 Circuit diagram of the project

12.1.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 12.6. The project is based on the PIC18F45K22 microcontroller, operating with an 8 MHz crystal (although most other types of PIC microcontrollers can also be used).

The connections between the GLCD and the microcontroller are as follows:

GLCD Pin	Microcontroller Pin
CS1	RB0
CS2	RB1
RS	RB2
R/W	RB3
EN	RB4
RST	RB5
D0	RD0
D1	RD1
D2	RD2

www.it-ebooks.info

D3	RD3
D4	RD4
D5	RD5
D6	RD6
D7	RD7

The contrast of the LCD is controlled by connecting a 10 KB potentiometer to pin Vo and Vee of the GLCD. The microcontroller is Reset using an external push-button.

If you are using the EasyPIC 7 development board, you should set the BCK jumper of SW4 to the +5 V position to enable the GLCD backlight. The GLCD contrast can be adjusted by the potentiometer located next to the display.

12.1.4 Project PDL

The PDL of this project is very simple and is given in Figure 12.7.

12.1.5 Project Program

The program is named GLCD1.C and the program listing of the project is shown in Figure 12.8. At the beginning of the program, the bitmap array is declared using the code generated by the Bitmap Editor. Then, the connection between the microcontroller and the GLCD are defined using *sbit* statements. The GLCD is connected to ports B and D of the microcontroller and thus both of these ports are configured as digital I/O ports using ANSEL statements (different PIC microcontrollers may require different settings). The GLCD library is then initialised using the Glcd_Init function. This function must be called before calling to any other GLCD function. The GLCD screen is then cleared using the Glcd_Fill(0×0), which turns OFF all pixels of the GLCD. The created bitmap is then displayed on the GLCD by calling to function Glcd_Image and entering the name of the bitmap array as an argument.

Figure 12.9 shows the created image displayed on the GLCD.

```
BEGIN
        Include bitmap character array
        Configure PORT B and PORT D as digital
        Configure PORT B and PORT D as outputs
        Initialise GLCD
        Clear GLCD screen
        Display bitmap image
END
```

Figure 12.7 PDL of the project

www.it-ebooks.info

```
/*************************************************************************************
                DISPLAYING GLCD BITMAP IMAGE
                ============================

This project shows how a bitmap image can be displayed on the GLCD. The bitmap image is
assumed to be in a file with extension BMP. The image file can be an existing bitmap file, or a
program (e.g. Windows Paint) can be used to create a new bitmap image and then save it as
a BMP file.

The GLCD Bitmap Editor is used first to convert the image to a character array.  mikroC GLCD
function Glcd_Image is then used to display the image on the GLCD.

It is important that the image must be monochromatic.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin       Microcontroller pin
   CS1            RB0
   CS2            RB1
   RS             RB2
   R/W            RB3
   RST            RB4
   EN             RB5
   D0 - D7        RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    GLCD1.C
*************************************************************************************/

// -------------------------------------------------
// GLCD Picture name: Untitled.bmp
// GLCD Model: KS0108 128x64
// -------------------------------------------------

const code char Face[1024] = {
  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 0, 128, 128, 128,
128, 128, 128, 128, 128, 128, 128, 128, 128,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
```

Figure 12.8 Program listing of the project

www.it-ebooks.info

```
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0, 128,  64,  32,  32,  16,  8,  8,  4,  4,  2,  2,  2,  1,  1,  1,  1,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  2,  2,  2,  4,  4,  8,  8,  16,  32,  32,  64, 128,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0, 224,  16,  12,  2,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  6,  8,  48,
192,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 248,  4,
248, 252,  3,  0,  0,  0,  0,  0,  0,  0, 224,  16,  8,  8,  16, 224,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0, 252,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 112, 136,  4,  4, 136, 112,  0,  0,  0,  0,
0,  0,  0,  0,  7, 248, 248,  4, 248,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,  4,
3,  7,  56, 192,  0,  0,  0,  0,  0,  0,  0,  1,  2,  2,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  3, 252,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,
192,  56,  7,  3,  4,  3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  1,  2,  12,  16,  32,  64, 128, 128,  0,  0,  0,  0,  0,  0, 192, 192, 192, 192,  32,  32,  32,  32,
32,  32,  32,  32,  32,  35,  32,  32,  32,  32,  32,  32,  32,  32,  32, 160, 192, 192, 192,  0,  0,  0,  0,  0,
0, 128,  64,  32,  16,  12,  3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  2,  4,  4,  8,  8,  16,  16,  16,  32,  33,  33,  33,  33,  65,  65,
65,  65,  65,  65,  65,  65,  65,  33,  33,  33,  33,  33,  16,  16,  16,  8,  8,  4,  4,  2,  1,  1,  0,  0,
0, 252,  32,  32,  32, 252,  0,  0, 224,  80,  80, 112,  0,  0, 252,  0,  0, 252,  0,  0, 176,  16,  16, 176,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  1,
0,  0,  1,  1,  1,  1,  0,  0,  1,  0,  0,  1,  0,  0,  1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0
};

// Glcd module connections
charGLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
```

Figure 12.8 (*Continued*)

www.it-ebooks.info

```
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections



//
// Start of main program
//
void main()
{
  ANSELB = 0;                                   // Configure PORT A as digital
  ANSELD = 0;                                   // Configure PORT B as digital
  TRISB = 0;                                    // PORT B is output
  TRISA = 0;                                    // PORT A is output

  Glcd_Init();                                  // Initialise GLCD
  Glcd_Fill(0x0);                               // Clear GLCD
  Glcd_Image(Face);                             // Draw the bitmap image

  while(1);                                     // End of program, wait here forever

}
```

Figure 12.8 (*Continued*)

Figure 12.9 Created image displayed on the GLCD

12.2 PROJECT 12.2 – Moving Ball Animation

12.2.1 Project Description

This project shows how the simple animation of a moving ball can be created on the GLCD. A filled circle is drawn to represent the ball. This figure is then moved across the GLCD screen from left-to-right and then right-to-left inside a rectangular shape. The text 'Moving

www.it-ebooks.info

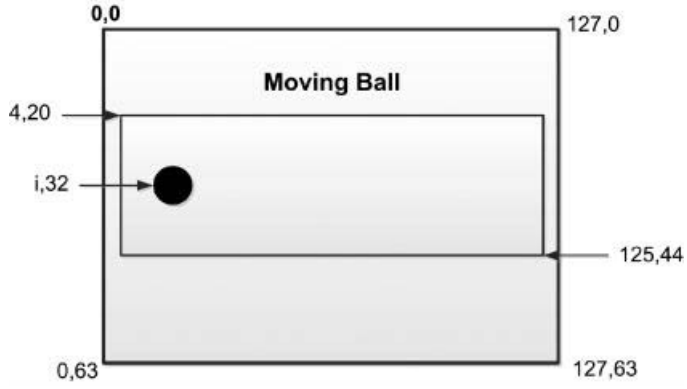


Figure 12.10 Ball animation

Ball' is written on the shape. The format of the display and co-ordinates of the shapes on the screen are shown in Figure 12.10.

12.2.2 Block Diagram

The block diagram of the project is as shown in Figure 12.5.

12.2.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 12.6.

12.2.4 Project PDL

The PDL of this project is given in Figure 12.11.

12.2.5 Project Program

The program is named GLCD2.C and the program listing of the project is given in Figure 12.12. At the beginning of the program, the connection between the microcontroller and the GLCD are defined using *sbit* statements. The GLCD is connected to ports B and D of the microcontroller and thus both of these ports are configured as digital I/O ports using ANSEL statements. The GLCD library is then initialised using the Glcd_Init function. This function must be called before calling to any other GLCD function. The GLCD screen is then cleared using the Glcd_Fill(0×0), which turns OFF all pixels of the GLCD.

The text 'Moving Ball' is displayed in Page 1, starting from x co-ordinate 30. Then a rectangle is drawn where the ball moves. The top left and bottom right co-ordinates of this rectangle are (4,20) and (125,44), respectively. Variable *flag* is used to determine the direction of movement. The ball moves from left-to-right when flag = 1, and from right-to-left when flag = 0. GLCD function Glcd_Circle_Fill is used to represent the ball. This function

www.it-ebooks.info

```
BEGIN
    Define the connection between the GLCD and the microcontroller
    Configure PORT A and PORT D as digital
    Configure PORT A and PORT D as output
    Initialise GLCD
    Clear GLCD screen
    Write text "Moving Ball"
    Draw a rectangle at (4,20), (125,44)
    WHILE x co-ordinate < largest x co-ordinate
        Draw a filled circle at (x,32)
        Wait for a while
        Delete the circle at (x,32)
        IF left-to-right movement THEN
            Increment x co-ordinate
        ELSE
            Decrement x co-ordinate
        ENDIF
        IF at furthest right point THEN
            Change direction of movement to right-to-left
        ENDIF
        IF at furthest left point THEN
            Change direction of movement to left-to-right
        ENDIF
END
```

Figure 12.11 PDL of the project

draws a filled circle with the specified radius, at the given x,y co-ordinates of the screen. The y co-ordinate of the ball is fixed at 32, while the x co-ordinate varies between 10 and 120 as the ball moves from left-to-right. The movement of the ball is animated first by displaying the ball, and after a short delay by erasing the ball and drawing a new one slightly to the right (or left) of the previous position. This way the ball shape seems as if it is moving. By varying the delay, we can modify the speed of movement. The x co-ordinate of the ball is incremented or decremented by 10 pixels as the ball moves from left-to-right or right-to-left, respectively. When the ball reaches the furthest point on the right-hand side, its direction is changed by clearing *flag* to 0. Similarly, when the ball reaches the furthest point on the left-hand side, its direction is changed by setting *flag* to 1.

The reader can experiment the effects on animation of changing the delay, size of the shape, and the step size.

Figure 12.13 shows the created image displayed on the GLCD screen.

12.3 PROJECT 12.3 – GLCD Dice

12.3.1 Project Description

In this project we design a GLCD based dice. The shapes of the numbers on two dices are imitated on a GLCD. The user presses a STRT button to 'throw' two dices. The dice numbers are then displayed for 5 seconds on the GLCD as real dice faces. After this time, the screen goes blank and the user is ready to 'throw' two new dices.

www.it-ebooks.info

```
/*************************************************************************************************
                BALL MOVEMENT ANIMATION
                ========================

This project shows how a simple animation can be created on the GLCD. Here, a filled circle
is displayed to represent a small ball. Also, a rectangle is drawn to show the boundaries of the
movement. The ball moves from left-to-right until it reaches the furthest point on the right.
Then, it moves from right-to-left until it reaches the furthest point on the left. This process is
repeated forever.

The speed of the ball is determined by including delay between the movements and the
speed can be changed by changing this delay.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is  connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin       Microcontroller pin
   CS1            RB0
   CS2            RB1
   RS             RB2
   R/W            RB3
   RST            RB4
   EN             RB5
   D0 - D7        RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee
and +5V supply.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    GLCD2.C
*************************************************************************************************/

// Glcd module connections
charGLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
```

Figure 12.12 Program listing of the project

www.it-ebooks.info

```
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections



//
// Start of main program
//
void main()
{
  unsigned char x,flag;
  ANSELB = 0;                                  // Configure PORT A as digital
  ANSELD = 0;                                  // Configure PORT B as digital
  TRISB = 0;                                   // PORT B is output
  TRISA = 0;                                   // PORT A is output

  Glcd_Init();                                 // Initialise GLCD
  Glcd_Fill(0x0);                              // Clear GLCD

  x = 10;                                      // Initial x co-ordinate and x step
  flag = 1;                                    // Start with left-to-right
  Glcd_Write_Text("Moving Ball",30,1,1);       // Write text
  Glcd_Rectangle(4,20,125,44,1);               // Draw the rectangle

  while(x < 127)
  {
    Glcd_Circle_Fill(x, 32, 4, 1);             // Draw the ball
    Delay_Ms(300);                             // Wait 300ms
    Glcd_Circle_Fill(x, 32, 4, 0);             // Erase the ball
    if(flag == 1)                              // If left-to-right
      x = x + 10;                              // Move ball right
    else                                       // Otherwise (right-to-left)
      x = x - 10;                              // Move ball left
    if(x == 120)flag = 0;                      // If at furthest right, change direction
    if(x == 10)flag = 1;                       // If at furthest left, change direction
  }

}
```

Figure 12.12 (*Continued*)

12.3.2 Block Diagram

The block diagram of the project is shown in Figure 12.14.

12.3.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 12.15. The GLCD is connected to PORT B and PORT D of the microcontroller. The STRT button is normally at logic 1 and is connected to port pin RC0 of the microcontroller. When the button is pressed, RC0 goes to logic 0.

www.it-ebooks.info

Figure 12.13 Created image displayed on the GLCD

12.3.4 Project PDL

The PDL of this project is given in Figure 12.16.

12.3.5 Project Program

The screen layout of the two dice shapes are designed using the Windows Paint program. Circles are used to represent the dots on a real dice. A dice number is shown by filling these circles to correspond to the dots on the faces of a real dice. Figure 12.17 shows the co-ordinates of the circles for the two dices. As an example, Figure 12.18 shows the dice numbers 2 and 5 being displayed on the GLCD.

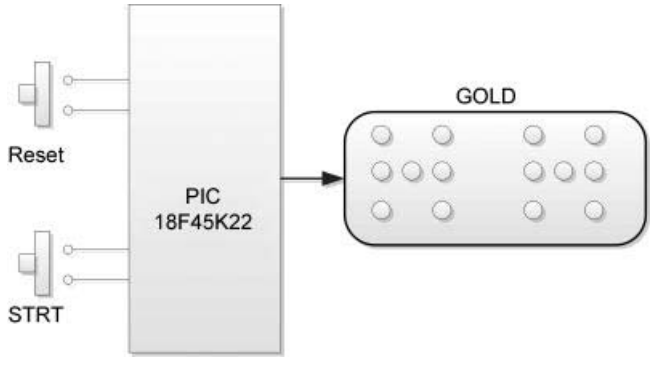


Figure 12.14 Block diagram of the project

www.it-ebooks.info

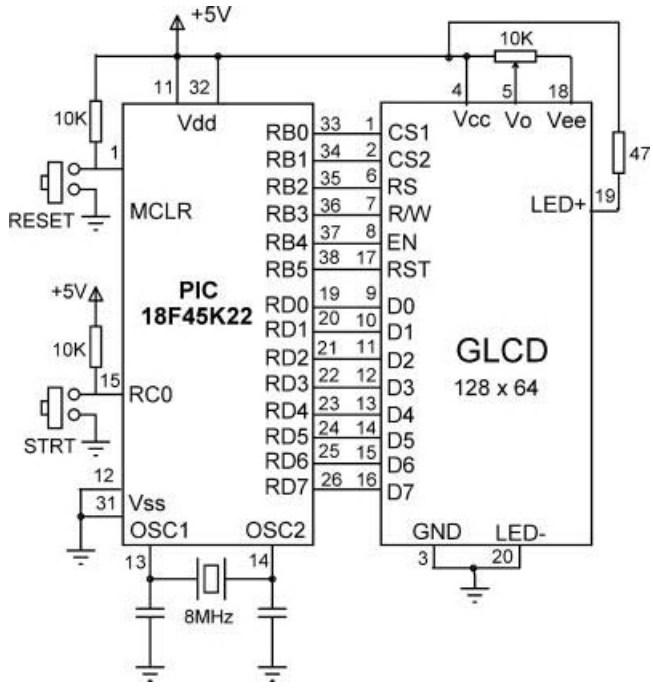


Figure 12.15 Circuit diagram of the project

The two dice x co-ordinates are separated from each other by 65 pixels. The radius of the dice circles are selected as 4 pixels.

The program is named GLCD3.C and Figure 12.19 shows the program listing of the project. At the beginning of the program, the radius of the circles is defined as 4 pixels and the STRT button is defined as connected to pin RC0 of the microcontroller. Then, the connection between the microcontroller and the GLCD are defined using *sbit* statements. The GLCD is connected to ports B and D of the microcontroller and thus both of these ports are configured as digital I/O ports using ANSEL statements. The GLCD library is then initialised using the Glcd_Init function. This function must be called before calling to any other GLCD function. The GLCD screen is then cleared using the Glcd_Fill(0×0), which turns OFF all pixels of the GLCD.

Function DisplayBackground is then called. This function displays 5 empty circles to imitate the faces of a real dice. Two dice faces are drawn, separated from each other by 65 pixels. The co-ordinates of the circles are as in Figure 12.17. When variable offset = 0, the first dice face is drawn, and when offset = 65, the second dice face is drawn.

The program then enters an endless loop, formed using a *for* statement. Inside this loop, the background is displayed and two random dice numbers (*Num1* and *Num2*) are generated between 1 and 6 using function RandomNumber. Function DisplayDice is then called to fill in the appropriate circles, so that the required number is displayed as the dots on the faces of real dices. Here, *N* is the number to be displayed, and *offset* determines whether the first or the second dice number is to be displayed. When *offset* is 0, the first dice number is displayed, and when *offset* is 65, the second dice number is displayed. *radius*

www.it-ebooks.info

```
BEGIN
    Define connection between GLCD and microcontroller
    Configure PORT B, PORT C, and PORT D as digital
    Configure PORT B and PORT D as output
    Configure RC0 as input
    Initialise GLCD
    Clear GLCD
    DO FOREVER
        CALL DisplayBackground
        Wait until STRT is pressed
        CALL RandomNumber to get the first dice number
        CALL RandomNumber to get the second dice number
        CALL DisplayDice to fill in circles to represent dots on a real dice for first number
        CALL DisplayDice to fill in circles to represent dots on a real dice for second number
        Wait 5 seconds
        CALL DisplayDice to empty the circle which has been filled in for first number
        CALL DisplayDice to empty the circle which has been filled in for second number
    ENDDO
END


BEGIN/RandomNumber
    Generate a number between 1 and 6
    Return the number to the calling program
END/RandomNumber


BEGIN/DisplayBackground
    Draw empty circles to represent the faces of two dices
END/DisplayBackground


BEGIN/DisplayDice
    IF offset = 0 THEN
        IF mode = 1 THEN
            Fill in the circles to represent faces of a real dice for first number
        ELSE
            Empty the circles for first number
        ENDIF
    ELSE IF offset = 65 THEN
        IF mode = 1 THEN
            Fill in the circles to represent faces of a real dice for second number
        ELSE
            Empty the circles for second number
        ENDIF
    ENDIF
END/DisplayDice
```

Figure 12.16 PDL of the project

is the radius of the circles in pixels. *mode* determines whether a circle should be filled in or not. When *mode* is 1, the circle is filled in to display a dot. When *mode* is 0, the circle is drawn blank. This mode is used when it is required to clear a dice number (i.e. to clear the circles). Function DisplayDice uses a *switch* statement to fill in or empty the circles based on the dice numbers.

www.it-ebooks.info

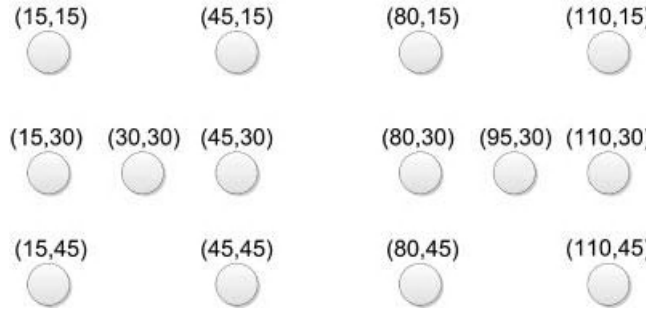


Figure 12.17 Co-ordinates of the dice circles

The program then waits for 5 seconds and then clears the circles by setting *mode* to 0 for the circles, which are filled in. At this point, the program is ready, and waits as above for the STRT button to be pressed again so that it generates new dice numbers.

Figure 12.20 shows a typical display of the GLCD.

12.3.6 Modifying the Program

The program given in Figure 12.19 can be made more user friendly by adding text to the display. For example, the text 'Start . . . ' can be added to the display before the user presses the STRT button. Also, the text 'Good Luck' can be added after the dice figures are displayed. The actual dice numbers can also be displayed at the bottom of each dice figure to make the display more user friendly.

Figure 12.21 shows the modified program (GLCD4.C). Here, the text 'Start . . . ' is displayed at the top of the display in Page 0, starting from x co-ordinate 40. The text 'Good Luck' is displayed at the same co-ordinates after the two dice numbers are generated. In addition, the two dice numbers *Num1* and *Num2* are converted into characters *n1* and *n2*, respectively. These characters are then displayed at the bottom of the dice figures. After displaying the texts and the dice figures, the program waits for 5 seconds. After this delay, the filled-in dice circles are changed into empty circles. In addition, the dice number texts are deleted and the text 'Good Luck' is changed to 'Start . . . ' to tell the user that the STRT button can be pressed again to generate two new dice numbers.

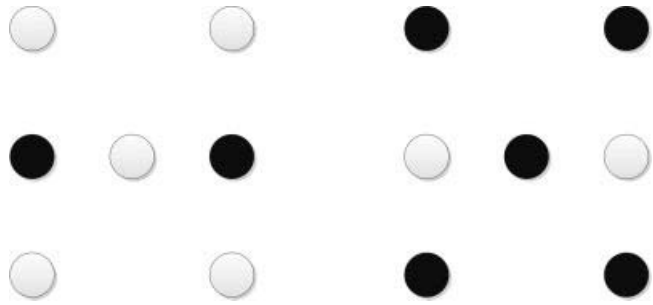


Figure 12.18 Dice numbers 2 and 5 displayed

www.it-ebooks.info

```
/*****************************************************************************************************
                                    GLCD DICE
                                    =========

This project shows how two dice faces can be imitated on a GLCD. A STRT button is used in the
project. The program waits until the STRT button is pressed. When the button is pressed, two
random numbers are generated between 1 and 6 and these numbers are shown on the GLCD
GLCD in the form of dice faces. The dots on a real dice are imitated with circles on the GLCD.
Circles are filled-in to represent the dice number thrown.

The two dice faces are organised as follows on the GLCD:

  o   o     o   o
  o o o     o o o
  o   o     o   o

The x co-ordinates of the two dices are separated by 65 pixels. The radius of each circle is
chosen as 4 pixels.

The dice numbers are displayed for 5 seconds, and after this time the GLCD screen is cleared
to indicate that the user can press the STRT button to create two new numbers.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin       Microcontroller pin
   CS1            RB0
   CS2            RB1
   RS             RB2
   R/W            RB3
   RST            RB4
   EN             RB5
   D0 - D7        RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    GLCD3.C
*****************************************************************************************************/
unsignedconst char radius = 4;                          // Radius of the circles

sbit STRT at RC0_bit;                                   // STRT button at RC0

// Glcd module connections
```

Figure 12.19 Program listing of the project

www.it-ebooks.info

```
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

//
// This function generates a pseudorandon integer number between 1 and Lim. A seed
// is given to the generator to start with
//
unsigned char RandomNumber(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
 }

//
// This function displays the background. The background consists of empty circles, organised
// as the faces of two real dices. When “offset” is 0, the first dice is drawn, and when offset is
// 65, the second dice face is drawn.
//
voidDisplayBackground(void)
{
  unsigned char offset;

  offset = 0;
  do
  {
    Glcd_Circle(15 + offset, 15, radius, 1);
    Glcd_Circle(45 + offset, 15, radius, 1);
    Glcd_Circle(15 + offset, 30, radius, 1);
    Glcd_Circle(30 + offset, 30, radius, 1);
    Glcd_Circle(45 + offset, 30, radius, 1);
    Glcd_Circle(15 + offset, 45, radius, 1);
    Glcd_Circle(45 + offset, 45, radius, 1);
    offset = offset + 65;
```

Figure 12.19 (*Continued*)

www.it-ebooks.info

```
  }while(offset == 65);
}


//
// This function fills the appropriate circle so that the required dice number is displayed as
// the faces of real dices. "N" is the number to be displayed, "offset"  determines whether
// the first or the second dice number is to be displayed. When "offset" is 0, the first dice
// number is displayed, and when "offset" is 65, the second dice number is displayed.
// "radius" is the radius of the circles in pixels. "mode" determines whether the circle should
// be filled in or not. When "mode" is 1 the circle is filled in to display a dot. When "mode" is
// 0, the circle is blank. This mode is used when it is required to clear a dice number (i.e. to
// clear the dots)
//
void DisplayDice(char N, char offset, char mode)
{
  switch(N)
  {
    case 1:                                                  // Dice number 1
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      break;
    case 2:                                                  // Dice number 2
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
      break;
    case 3:                                                  // Dice number 3
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
      break;
    case 4:                                                  // Dice number 4
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 45, radius, mode);
      Glcd_Circle_Fill(45 + offset, 45, radius, mode);
      break;
    case 5:                                                  // Dice number 5
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 45, radius, mode);
      Glcd_Circle_Fill(45 + offset, 45, radius, mode);
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      break;
    case 6:                                                  // Dice number 6
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
      Glcd_Circle_Fill(15 + offset, 45, radius, mode);
      Glcd_Circle_Fill(45 + offset, 45, radius, mode);
      break;
```

Figure 12.19 (*Continued*)

www.it-ebooks.info

```
  }
}


//
// Start of main program
//
void main()
{
  unsigned char Num1, Num2;
  unsigned char seed = 1;

  ANSELB = 0;                          // Configure PORT B as digital
  ANSELC = 0;                          // Configure PORT C as digital
  ANSELD = 0;                          // Configure PORT D as digital
  TRISB = 0;                           // PORT B is output
  TRISC = 1;                           // RC0 is input

  Glcd_Init();                         // Initialise GLCD
  Glcd_Fill(0x0);                      // Clear GLCD

  for(;;)                              // DO FOREVER
  {
    DisplayBackground();               // Display background (empty circles)
    while(STRT == 1);                  // Wait until STRT is pressed

    Num1 = RandomNumber(6, seed);      // Generate first dice number
    Num2 = RandomNumber(6, seed);      // Generate second dice number
    DisplayDice(Num1, 0, 1);           // Display first dice number
    DisplayDice(Num2, 65, 1);          // Display second dice number
    Delay_Ms(5000);                    // Wait 5 seconds
    DisplayDice(Num1, 0, 0);           // Empty the circles for first dice
    DisplayDice(Num2, 65, 0);          // Empty the circles for second dice
  }
}
```

Figure 12.19 (*Continued*)

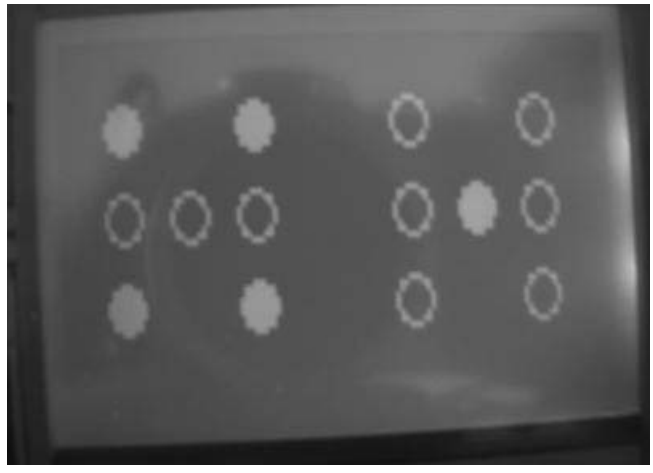


Figure 12.20 A typical display

www.it-ebooks.info

```
/*************************************************************************************************************
                                    GLCD DICE
                                    =========

This project shows how two dice faces can be imitated on a GLCD. A STRT button is used in the
project. The program waits until the STRT button is pressed. When the button is pressed, two
random numbers are generated between 1 and 6 and these numbers are shown on the GLCD
GLCD in the form of dice faces. The dots on a real dice are imitated with circles on the GLCD.
Circles are filled-in to represent the dice number thrown.

The two dice faces are organised as follows on the GLCD:

   o   o    o   o
   o o o    o o o
   o   o    o   o

The x co-ordinates of the two dices are separated by 65 pixels. The radius of each circle is
chosen as 4 pixels.

The dice numbers are displayed for 5 seconds, and after this time the GLCD screen is cleared
to indicate that the user can press the STRT button to create two new numbers.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin      Microcontroller pin
   CS1           RB0
   CS2           RB1
   RS            RB2
   R/W           RB3
   RST           RB4
   EN            RB5
   D0 - D7       RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.

In this modified version of the program texts are added to the display to make it more user
Friendly.


Author:    Dogan Ibrahim
Date:      December, 2011
File:      GLCD4.C
*************************************************************************************************************/
unsignedconst char radius = 4;                          // Radius of the circles
```

Figure 12.21 Program listing

www.it-ebooks.info

```
sbit STRT at RC0_bit;                                   // STRT button at RC0

// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

//
// This function generates a pseudorandon integer number between 1 and Lim. A seed
// is given to the generator to start with
//
unsigned char RandomNumber(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
 }

//
// This function displays the background. The background consists of empty circles, organised
// as the faces of two real dices. When "offset" is 0, the first dice is drawn, and when offset is
// 65, the second dice face is drawn.
//
voidDisplayBackground(void)
{
  unsigned char offset;

  offset = 0;
  do
  {
    Glcd_Circle(15 + offset, 15, radius, 1);
    Glcd_Circle(45 + offset, 15, radius, 1);
    Glcd_Circle(15 + offset, 30, radius, 1);
    Glcd_Circle(30 + offset, 30, radius, 1);
    Glcd_Circle(45 + offset, 30, radius, 1);
```

Figure 12.21 (*Continued*)

www.it-ebooks.info

```
    Glcd_Circle(15 + offset, 45, radius, 1);
    Glcd_Circle(45 + offset, 45, radius, 1);
    offset = offset + 65;
  }while(offset == 65);
}


//
// This function fills the appropriate circle so that the required dice number is displayed as
// the faces of real dices. “N” is the number to be displayed, “offset”  determines whether
// the first or the second dice number is to be displayed. When “offset” is 0, the first dice
// number is displayed, and when “offset” is 65, the second dice number is displayed.
// “radius” is the radius of the circles in pixels. “mode” determines whether the circle should
// be filled in or not. When “mode” is 1 the circle is filled in to display a dot. When “mode” is
// 0, the circle is blank. This mode is used when it is required to clear a dice number (i.e. to
// clear the dots)
//
voidDisplayDice(char N, char offset, char mode)
{
  switch(N)
  {
    case 1:                                                  // Dice number 1
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      break;
    case 2:                                                  // Dice number 2
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
      break;
    case 3:                                                  // Dice number 3
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
      break;
    case 4:                                                  // Dice number 4
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 45, radius, mode);
      Glcd_Circle_Fill(45 + offset, 45, radius, mode);
      break;
    case 5:                                                  // Dice number 5
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 45, radius, mode);
      Glcd_Circle_Fill(45 + offset, 45, radius, mode);
      Glcd_Circle_Fill(30 + offset, 30, radius, mode);
      break;
    case 6:                                                  // Dice number 6
      Glcd_Circle_Fill(15 + offset, 15, radius, mode);
      Glcd_Circle_Fill(45 + offset, 15, radius, mode);
      Glcd_Circle_Fill(15 + offset, 30, radius, mode);
      Glcd_Circle_Fill(45 + offset, 30, radius, mode);
```

Figure 12.21 (*Continued*)

www.it-ebooks.info

```
        Glcd_Circle_Fill(15 + offset, 45, radius, mode);
        Glcd_Circle_Fill(45 + offset, 45, radius, mode);
        break;
      }
  }


  //
  // Start of main program
  //
  void main()
  {
    unsigned char Num1, Num2, n1, n2;
    unsigned char seed = 1;

    ANSELB = 0;                                  // Configure PORT B as digital
    ANSELC = 0;                                  // Configure PORT C as digital
    ANSELD = 0;                                  // Configure PORT D as digital
    TRISB = 0;                                   // PORT B is output
    TRISC = 1;                                   // RC0 is input

    Glcd_Init();                                 // Initialise GLCD
    Glcd_Fill(0x0);                              // Clear GLCD

    for(;;)                                      // DO FOREVER
    {
      DisplayBackground();                       // Display background (empty circles)
      Glcd_Write_Text("Start...", 40, 0, 1);     // Display text before starting
      while(STRT == 1);                          // Wait until STRT is pressed

      Glcd_Write_Text("Good Luck", 40, 0, 1);    // Display text after starting
      Num1 = RandomNumber(6, seed);              // Generate first dice number
      Num2 = RandomNumber(6, seed);              // Generate second dice number
      DisplayDice(Num1, 0, 1);                   // Display first dice number
      DisplayDice(Num2, 65, 1);                  // Display second dice number
      n1 = Num1 + '0';                           // Convert to character
      n2 = Num2 + '0';                           // Convert to character
      Glcd_Write_Char(n1, 27, 7, 1);             // Display dice number under dice 1
      Glcd_Write_Char(n2, 92, 7, 1);             // Display dice number under dice 2
      Delay_Ms(5000);                             // Wait 5 seconds
      DisplayDice(Num1, 0, 0);                   // Empty the circles for first dice
      DisplayDice(Num2, 65, 0);                  // Empty the circles for second dice
      Glcd_Write_Text("         ", 40, 0, 1);    // Clear text “Good Luck”
      Glcd_Write_Char(' ', 27, 7, 1);            // Clear number under dice 1
      Glcd_Write_Char(' ', 92, 7, 1);            // Clear number under dice 2
    }
  }
```

Figure 12.21 (*Continued*)

Figure 12.22 shows the GLCD screen before the STRT button is pressed. Figure 12.23 shows the screen after the STRT button is pressed, when two new dice numbers are generated with the modified program.

Note: The modified program generates a hex code over 2 KB, and thus the Demo version of the mikroC Pro for PIC compiler cannot be used to compile this program.

www.it-ebooks.info

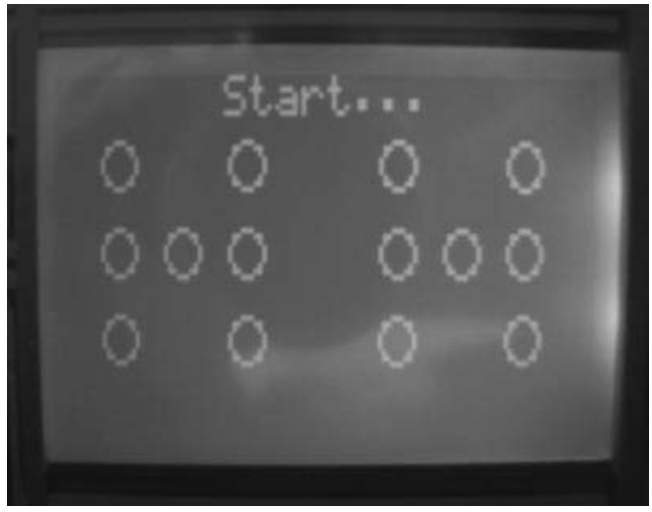


Figure 12.22 Before pressing the STRT button

Figure 12.23 After pressing the STRT button

12.4 PROJECT 12.4 – GLCD X-Y Plotting

12.4.1 Project Description

This project will demonstrate how the graph of a function can be plotted on the GLCD. In this project, the graph of $y = x^2 - 1$ is plotted as an example, with x varying between -2 and $+2$.

The mid-point of the GLCD screen is taken as the co-ordinate centre point of the X-Y axis, and the two axes are drawn from this point, as shown in Figure 12.24, that is the GLCD co-ordinate (63,31) is taken to be the (0,0) co-ordinate point of our axes. The X-axis is the horizontal axis extending from left to right. Similarly, the Y-axis is the vertical axis extending from bottom to top.

www.it-ebooks.info

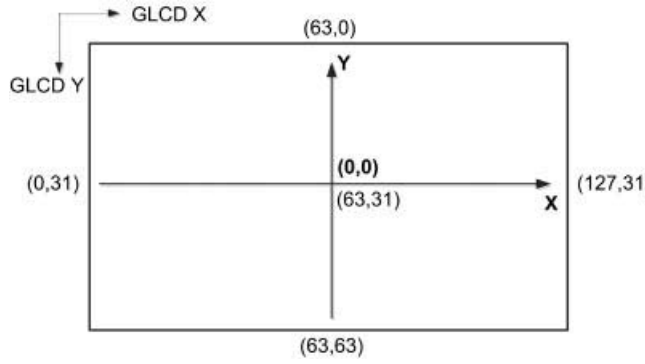


Figure 12.24 The GLCD screen co-ordinates and X-Y co-ordinates

12.4.2 Block Diagram

The block diagram of the project is as shown in Figure 12.5.

12.4.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 12.6.

12.4.4 Project PDL

The PDL of this project is given in Figure 12.25.

```
BEGIN
        Define connections between the GLCD and the microcontroller
        Configure PORT B and PORT D as digital
        Configure PORT B and PORT D as output
        Initialise GLCD
        Clear GLCD
        Store the X and corresponding Y values of the function in arrays
        CALLPlotXY to plot the graph
END


BEGIN/PlotXY
        Draw X and Y axes
        Draw tick points on axes
        Find the maximum and minimum X and Y values
        DO for all points
                Calculate screen X co-ordinate
                Calculate screen Y co-ordinate
                Plot the graph by enabling pixels at screen X and Y co-ordinates
        ENDDO
END/PlotXY
```

Figure 12.25 PDL of the project

www.it-ebooks.info

12.4.5 Project Program

Figure 12.26 shows the program listing (GLCD5.C) of the project. At the beginning of the program, the connection between the microcontroller and the GLCD are defined using *sbit* statements. The GLCD is connected to ports B and D of the microcontroller and thus both of these ports are configured as digital I/O ports using ANSEL statements. The GLCD library is then initialised using the Glcd_Init function. This function must be called before calling to any other GLCD function. The GLCD screen is then cleared using the Glcd_Fill(0×0), which turns OFF all pixels of the GLCD.

The starting and ending x values are stored in variables *starting_x* and *ending_x*, respectively. The program then calculates the Y values of the function for the given X values. A total of 100 points are considered in this example. The graph of the function is drawn by calling function PlotXY. The X and Y values of the function, and the total number of points are entered as the arguments of the function.

Function PlotXY is where the actual graph is drawn. First, the X and Y axes and the axis ticks are drawn using GLCD functions Glcd_Line and Glcd_Dot statements, respectively. The axes are drawn, as shown in Figure 12.24. The function then calculates the maximum and minimum X and Y values of the function, so that the screen co-ordinates can be scaled correctly. The actual graph plotting is done in a *for* loop, where the X and Y points are plotted as dots, using the GLCD function Glcd_Dot. The X and Y co-ordinates the points to be plotted are calculated as

$$\begin{aligned} \text{Xpos} &= 63.0 + \text{X[i]} * 63.0/\text{X max}; \\ \text{Ypos} &= 31.0 - \text{Y[i]} * 31.0/\text{Y max}; \end{aligned} \tag{12.1}$$

where Xpos and Ypos are the X and Y co-ordinates of points, respectively, arrays X[i] and Y[i] store the X and corresponding Y values, respectively, and Xmax and Ymax are the maximum X and Y values of all the points.

Figure 12.27 shows the graph of $y = x^2 - 1$ plotted on the GLCD.

12.5 PROJECT 12.5 – Plotting Temperature Variation on the GLCD

12.5.1 Project Description

This project demonstrates how the ambient temperature can be measured and then plotted in real-time on the GLCD. The temperature is measured every second using an LM35DZ type analogue sensor and is then plotted in real-time on the GLCD.

The *x* and *y* axes are drawn on the GLCD, the axes ticks are displayed, and the Y axis is labelled, as shown in Figure 12.28. The Y axis is the temperature, and the X axis is the time where every pixel corresponds to 1 second in real-time.

12.5.2 Block Diagram

The block diagram of the project is shown in Figure 12.29.

www.it-ebooks.info

```
/*************************************************************************************************************
                GLCD X-Y GRAPH PLOTTING
                ======================

This project shows how an X-Y graph can be plotted on the GLCD screen. In this project the
function of y = x*x - 1 is plotted within the region x = -2 and x = 2.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin       Microcontroller pin
   CS1            RB0
   CS2            RB1
   RS             RB2
   R/W            RB3
   RST            RB4
   EN             RB5
   D0 - D7        RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to pin
Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and +5V supply.


Author:   Dogan Ibrahim
Date:     December, 2011
File:     GLCD5.C
*************************************************************************************************************/

// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections


//
```

Figure 12.26 Program listing of the project

www.it-ebooks.info

```
// This function draws the actual graph on the LCD. First, the X and Y axis are drawn and
// the axes ticks are displayed. Then, the maximum and minimum values of the data
// points are calculated so that the screen can be scaled correctly. Then, the data points
// are plotted by placing “dots” at their co-ordinates
//
void PlotXY(float X[], float Y[], unsigned char N)
{
  unsigned char i;
  float Xmax, Ymax, Xpos, Ypos,Xmin,Ymin;

  Glcd_Line(63, 0, 63, 63, 1);                    // Draw Y axis
  Glcd_Line(0, 31, 127, 31, 1);                   // Draw X axis
  for(i=0; i<127; i += 9)Glcd_Dot(i, 32, 1);      // Display x axis ticks
  for(i=0; i<63; i += 9)Glcd_Dot(64, i, 1);       // Display y axis ticks
//
// Find the maximum values Xmax, Ymax and minimum values Xmin and Ymin
//
  Xmax = 0;                                       // Assume 0 to start with
  Ymax = 0;                                       // Assume 0 to start with
  Xmin=0;                                         // Assume 0 to start with
  Ymin=0;                                         // Assume 0 t start with

  for(i = 0; i<= N; i++)                          // Do for all points
  {
    if(X[i] >Xmax)Xmax = X[i];
    if(Y[i] >Ymax)Ymax = Y[i];
    if(X[i] <Xmin)Xmin = X[i];
    if(Y[i] <Ymin)Ymin = Y[i];
  }

  Xmax=fabs(Xmax);                                // Find the absolute value
  Ymax = fabs(Ymax);                              // Find the absolute value
  Xmin=fabs(Xmin);                                // Find the absolute value
  Ymin=fabs(Ymin);                                // Find the absolute value
  if(Xmax<Xmin)Xmax = Xmin;
  if(Ymax<Ymin)Ymax = Ymin;

//
// Now plot the graph. The graph is plotted by first calculating the screen co-ordinates
// of the X and Y points. Then, GLCD function Glcd_Dot is used to place a pixel at these
// data points
//
  for(i = 0; i<= N; i++)
  {
    Xpos = 63.0 + X[i]*63.0 / Xmax;               // Calculate x co-ordinates
    Ypos = 31.0 - Y[i]*31.0 / Ymax;               // Calculate y co-ordinates
    Glcd_Dot(Xpos, Ypos, 1);                      // Plot the graph
  }
}
```

Figure 12.26 (*Continued*)

12.5.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 12.30. The LM35DZ temperature sensor is connected to analogue port RA0 (or AN0) of the microcontroller. This is a 3-pin

www.it-ebooks.info

```
//
// Start of main program
//
void main()
{
  floatXvalues[101], Yvalues[101], stp, x, starting_x, ending_x;
  unsigned char N , i;

  ANSELB = 0;                                  // Configure PORT B as digital
  ANSELD = 0;                                  // Configure PORT D as digital
  TRISB = 0;                                   // PORT B is output
  TRISD = 0;                                   // PORT D is output

  Glcd_Init();                                 // Initialise GLCD
  Glcd_Fill(0x0);                              // Clear GLCD

//
// N is the total number of points considered. stp is the step in x value
//
  N = 100;                                     // No of points
  starting_x = -2.0;                           // Starting X value
  ending_x = 2.0;                              // Ending X value
  stp = fabs((starting_x - ending_x)) / N;     // Step in x

  x = starting_x;                              // Starting X value
  i = 0;
//
// Find the Y values of the function for given X values and store these values in arrays
// Yvalues and Xvalues respectively
//
  do
  {
    Xvalues[i] = x;                            // X values of the function
    Yvalues[i] = x*x -1;                       // Calculate Y yalues of the function
    x = x + stp;                               // step in x
    i++;                                       // Next point
  }while(i != N+1);                            // Do for all points

  PlotXY(Xvalues, Yvalues, N);                 // Plot the graph

  while(1);                                    // End of program, wait here forever
}
```

Figure 12.26 (*Continued*)

analogue temperature sensor integrated circuit. Two of the pins are connected to +5 V and ground, where the third pin is the output. This sensor provides an output voltage directly proportional to the measured temperature. The output of the sensor is given by

$$\text{Vo} = 10\,\text{mV}/^{\circ}\text{C} \tag{12.2}$$

Thus, for example at 10°C the output is 100 mV, at 25°C the output is 250 mV, and so on. PORT B and PORT D are connected to the GLCD as before.

www.it-ebooks.info

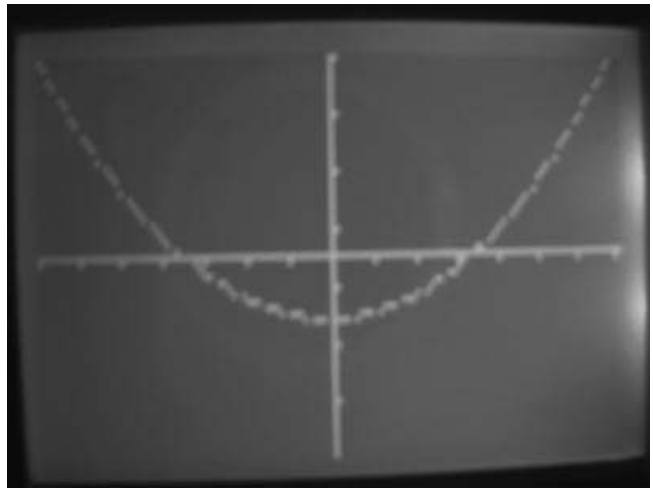


Figure 12.27 Graph of $y = x^2 - 1$ plotted on the screen

12.5.4 Project PDL

The PDL of this project is given in Figure 12.31.

12.5.5 Project Program

The A/D converter on the PIC18F45K22 microcontroller is 10-bits wide. Thus, with a +5 V reference voltage, the resolution will be 5000/1024 or 4.88 mV, which is accurate enough to measure the temperature to an accuracy of 0.5°C.

Figure 12.32 shows the program listing (GLCD6.C) of the project. At the beginning of the program, the connections between the microcontroller and the GLCD are defined

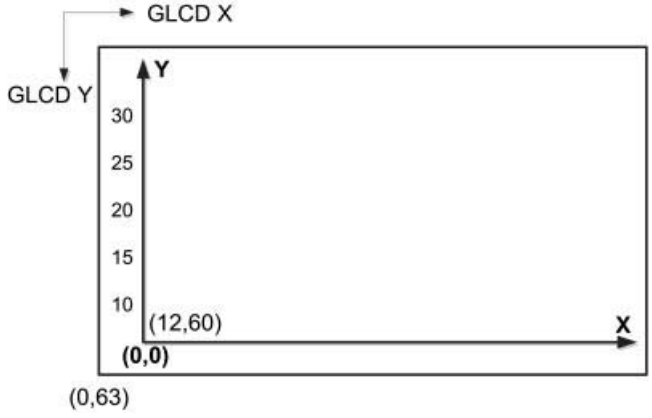


Figure 12.28 Layout of the screen

www.it-ebooks.info

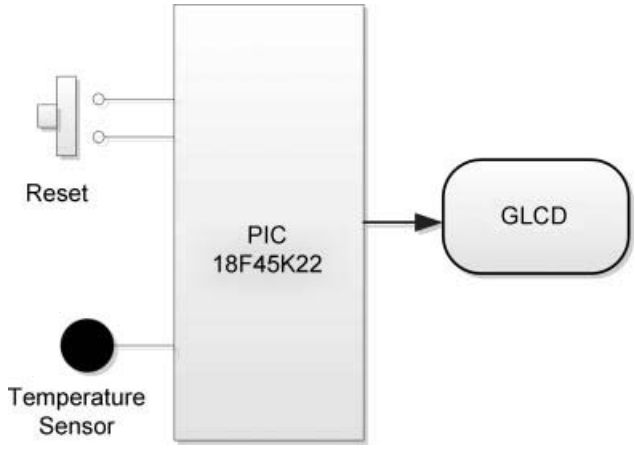


Figure 12.29 Block diagram of the project

using *sbit* statements. The GLCD is connected to ports B and D of the microcontroller and thus both of these ports are configured as digital output ports using ANSEL and TRIS statements. PORT A is configured as analogue, with pin RA0 (or AN0) being configured as an input.

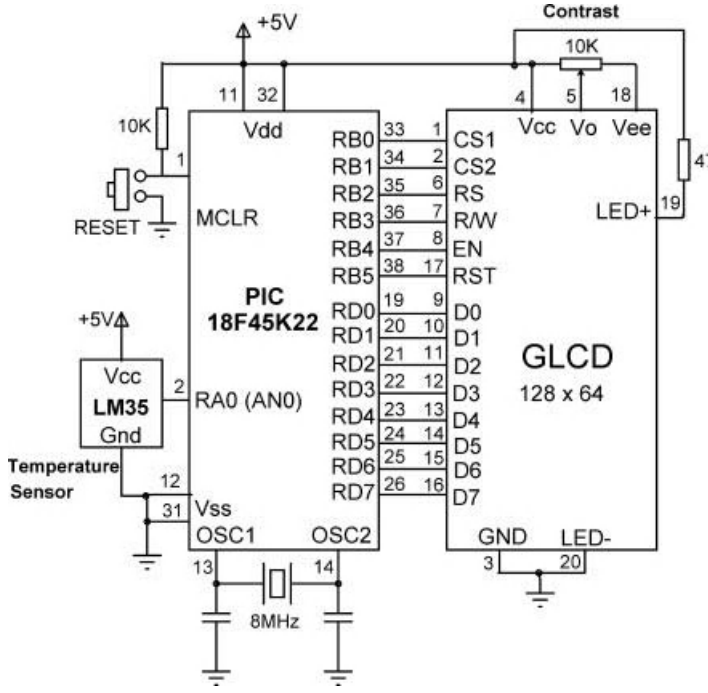


Figure 12.30 Circuit diagram of the project

www.it-ebooks.info

```
BEGIN
    Define the connection between the LCD and the microcontroller
    Configure PORT B and PORT D as digital output
    Configure PORT A as analog input
    Initialise GLCD
    Clear GLCD
    Initialise A/D converter
    CALL PlotAxis
    DO FOREVER
        Read analog temperature from Channel 0
        Convert into millivolts
        Convert into Degrees centigrade
        Calculate the Y co-ordinate based on temperature reading
        CALL PlotXY to plot the temperature
        Wait 1 second
    ENDDO
END

BEGIN/PlotAxis
    Draw X and Y axes
    Draw axes ticks
    Draw Y axis labels
END/PlotAxis

BEGIN/PlotXY
    Draw a line to join previous and current temperature values
    Update the previous X and Y values with current values
END/PlotXY
```

Figure 12.31 PDL of the project

The GLCD library is then initialised using the Glcd_Init function. This function must be called before calling to any other GLCD function. The GLCD screen is then cleared using the Glcd_Fill(0×0), which turns OFF all pixels of the GLCD.

The A/D converter is initialised by calling library function ADC_Init. The background of the display is drawn by calling function PlotAxis. This function draws the X and Y axes. The bottom left part of the screen with co-ordinates (12,60) is taken as the (0,0) co-ordinate of our display. Then, ticks are placed on both the X and the Y axes using Glcd_Dot statements. The Y axis is labelled from 10 to 30°C in steps of 5°C, using the Glcd_Write_Text_Adv statements.

The program then enters an endless loop formed by a *for* statement. Inside this loop, the analogue temperature is converted into digital and stored in variable *T* by calling function ADC_Get_Sample with the channel number specified as 0 (RA0 or AN0). This digital value is converted into millivolts by multiplying with 5000 and dividing by 1024. The actual temperature in °C is calculated by dividing the voltage in millivolts by 10 (Vo = 10 mV/°C).

The graph is drawn using the GLCD function Glcd_Line. This function draws a line between the specified starting and ending X and Y co-ordinates. Variables old_x, old_y, new_x, and new_y are used to store the old and the new (current) X and Y values of the temperature, respectively. At the first iteration, the old and the current values are assumed to

www.it-ebooks.info

```
/************************************************************************************************************
                GLCD TEMPERATURE PLOTTING IN REAL-TIME
                =====================================

This project shows how the temperature can be read from an analog temperature sensor and then
plotted on the GLCD.

In this project an LM35DZ type analog temperature sensor is used. This sensor has 3 pins: The
ground, power supply (+5V), and the output pin. The sensor gives an output voltage which is
directly proportional to the measured temperature. i.e. V o = 10mV/C. Thus, for example at
15C the output voltage is 150mV. Similarly, at 30C the output voltage is 300mV and so on.

The temperature sensor is connected to analog input RA0 (or AN0) of a PIC18F45K22 type
microcontroller. The microcontroller is operated from an 8MHz crystal. The GLCD used in the
project is based on KS0107/108 type controller with 128x64 pixels.

The program first draws the X and Y axes, axes ticks, and the Y axis labels. Then, the temperature
is read from Channel 0 (RA0 or AN0), converted into digital, and then into Degrees C. The
temperature is plotted in real-time every second. i.e. the horizontal axis is the time where each
pixel corresponds to 1 second.

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin      Microcontroller pin
   CS1           RB0
   CS2           RB1
   RS            RB2
   R/W           RB3
   RST           RB4
   EN            RB5
   D0 - D7       RD0 - RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.


Author:   Dogan Ibrahim
Date:     December, 2011
File:     GLCD6.C
************************************************************************************************************/

unsigned char stp, old_x, old_y, new_x, new_y;

// Glcd module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
```

Figure 12.32 Program listing of the project

www.it-ebooks.info

```
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections


//
// This function plots the X and Y axis. The origin is set at screen co-ordinates (12,60).
// First the two axes are drawn. Then the axes ticks are displayed for both X and Y axis.
// Finally, the Y axis labels are displayed (i.e. the temperature labels)
//
void PlotAxis()
{
  unsigned char i;

  Glcd_Line(12, 0, 12, 60, 1);                    // Draw Y axis
  Glcd_Line(12, 60, 127, 60, 1);                  // Draw X axis
  for(i=12; i<127; i += 9)Glcd_Dot(i, 61, 1);     // Display x axis ticks
  for(i=0; i<60; i += 10)Glcd_Dot(11, i, 1);      // Display y axis ticks
  Glcd_Write_Text_Adv("30",0,5);                  // Y axis label
  Glcd_Write_Text_Adv("25",0,15);                 // Y axis label
  Glcd_Write_Text_Adv("20",0,25);                 // Y axis label
  Glcd_Write_Text_Adv("15",0,35);                 // Y axis label
  Glcd_Write_Text_Adv("10",0,45);                 // Y axis label
}

//
// This function plots the temperature in real-time. The temperature is plotted by joining
// the data points with straight lines. The X axis is the time where each pixel corresponds
// to one second. The Y axis is the temperature in Degrees C
//
void PlotXY(float Temperature)
{
  Glcd_Line(old_x,old_y,new_x,new_y,1);           // Draw temperature changes
  old_x = new_x;                                  // Update old points
  old_y = new_y;
}


//
// Start of main program
//
void main()
{
  unsignedint T;
```

Figure 12.32 (*Continued*)

www.it-ebooks.info

```
unsigned char flag = 0;
float mV, C;

ANSELA = 1;                          // Configure PORT A as analog
ANSELB = 0;                          // Configure PORT B as digital
ANSELD = 0;                          // Configure PORT D as digital
TRISA = 1;                           // RA0 is input (analog)
TRISB = 0;                           // PORT B is output
TRISD = 0;                           // PORT D is output

Glcd_Init();                         // Initialise GLCD
Glcd_Fill(0x0);                      // Clear GLCD
ADC_Init();                          // Initialise ADC
PlotAxis();                          // Plot X-Y axes and labels

for(;;)                              // DO FOREVER
{
  T = ADC_Get_Sample(0);             // Read temperature from channel 0
  mV = T*5000.0/1024.0;              // Temperature in mV
  C = mV /10.0;                      // Temperature in Degrees C

  if(flag == 0)                      // If first time
  {
    new_x = 12;                      // Start from x = 12
    old_x = new_x;
    new_y = -2*C+70;                 // New temperature value
    old_y = new_y;
    flag = 1;                        // Set so that not first time
  }
  else                               // Not first time
  {
    new_x++;                         // Increment x by 1 (1 second each pixel)
    new_y = -2*C+70;                 // New temperature value
  }
  PlotXY(C);                         // Plot the graph
  Delay_Ms(1000);                    // Wait one second
  }
}
```

Figure 12.32 (*Continued*)

be the same and this is identified by the variable *flag* being cleared to 0. In all other iterations, variable *flag* is 1 and the *else* part of the *if* statement is executed. The X value is incremented by 1 to correspond to the next second and the new Y value is updated.

The Y co-ordinate (temperature) is calculated as follows:

The relationship between the Y axis ticks and the Y co-ordinates of data values can be derived from the following:

Y axis ticks pixel co-ordinates	Y axis data co-ordinate (°C)
10	30
20	25
30	20
40	15
50	10

www.it-ebooks.info

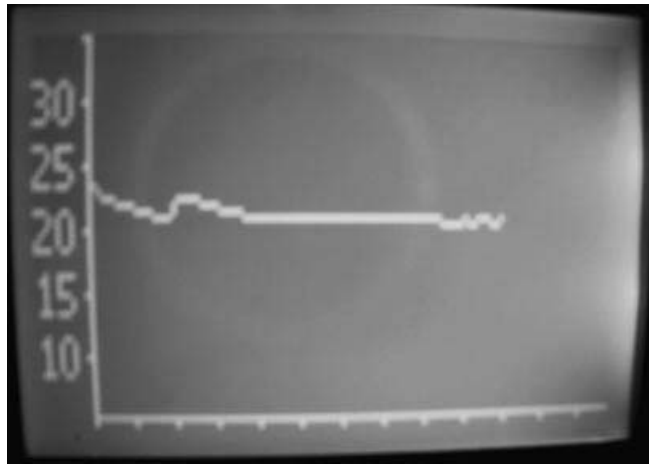


Figure 12.33 Typical display of the temperature

The above relationship is linear and is the form of a straight line $y = mx + C$, where m is the slope of the line and C is the point where the line crosses the Y axis. The equation of this line can be found from

$$y - y_1 = m(x - x_1) \tag{12.3}$$

where $m = (y_2 - y_1)/(x_2 - x_1)$

By taking any two points on the line, we can easily find the equation. Considering the points:

$$(x_1, y_1) = (30, 10) \text{ and } (x_2, y_2) = (10, 50) \tag{12.4}$$

The relationship is found to be:

$$y = -2x + 70 \tag{12.5}$$

Therefore, given the temperature C in degrees, the y co-ordinate to be used for plotting can be calculated from

$$\text{new\_y} = -2^{*}C + 70 \tag{12.6}$$

After plotting a point, the *new_x* and *new_y* are copied to *old_x* and *old_y*, respectively, ready for the next sample to be plotted.

Figure 12.33 shows a typical display of the temperature in real-time.

12.5.6 Suggestion for Modification

The temperature measurement can be made more accurate by using a lower reference voltage for the A/D converter. For example, if a +3 V reference voltage is used instead of the +5 V,

www.it-ebooks.info

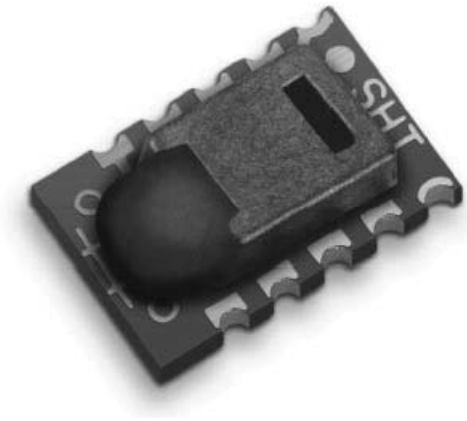


Figure 12.34 The SHT11 sensor

then the resolution of the A/D converter will be 3000/1024 = 2.92 mV instead of the 4.88 mV used in the project.

12.6 PROJECT 12.6 – Temperature and Relative Humidity Measurement

12.6.1 Project Description

This project demonstrates how the ambient temperature and relative humidity can be measured and then displayed on the GLCD.

In this project, the SHT11 relative humidity and temperature sensor chip is used. This is a tiny 8-pin chip with dimensions 4.93 × 7.47 mm and thickness 2.6 mm, manufactured by Sensirion (http://www.sensirion.com). A capacitive sensor element is used to measure the relative humidity, while the temperature is measured by a band-gap sensor. A calibrated digital output is given for ease of connection to a microcontroller. The relative humidity is measured with an accuracy of ±4.5%RH and the temperature accuracy is ±0.5°C. Operating voltage ranges from a minimum of +2.4 V to a maximum of +5.5 V.

Figure 12.34 shows a picture of the SHT11 sensor. The sensor is available as a small PCB that can be plugged into a development board (e.g. EasyPIC 7) for ease of use.

The pin configuration of the sensor is as follows:

Pin 1: *Gnd pin*.
Pin 2: *The Data pin*. This pin is used to transfer data in and out of the sensor. When sensing a command to the sensor, the data is valid on the rising edge of the clock input (SCK). When reading data from the sensor, data is valid 200 ns after the clock goes low and remains valid until the next clock.
Pin 3: *The Clock pin*. This pin is used to synchronise the chip with the microcontroller during command and data transfers.
Pin 4: *VDD pin*.

www.it-ebooks.info

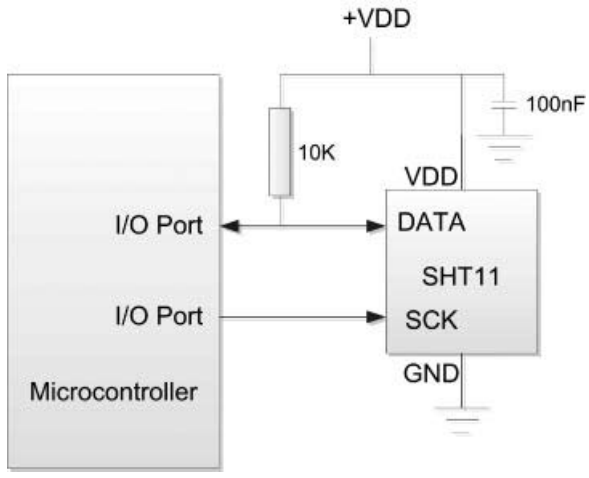


Figure 12.35 Connection of SHT11 to a microcontroller

As shown in Figure 12.35, the SHT11 sensor is connected to a microcontroller using only two pins. The data pin should be pulled up to the supply voltage using a 10 KB resistor. It is recommended to use a 100 nF decoupling capacitor between pin 4 and ground.

12.7 Operation of the SHT11

The SHT11 is based on serial communication where data is clocked in and out, in synchronisation with the SCK clock. The communication between the SHT11 and a microcontroller consists of the following protocols (see the SHT11 data sheet for more detailed information).

12.7.1 RESET

At the beginning of data transmission, it is recommended to send a RESET to the SHT11, just in case the communication with the device is lost. This signal consists of sending 9 or more SCK signals while the DATA line is HIGH. Then, a Transmission-Start-Sequence must be sent. Figure 12.36 shows the RESET sequence.

The C code to implement the RESET sequence as a function is given below (SDA and SCK are the DATA and SCK lines, respectively). Notice that the manufacturer's data sheet

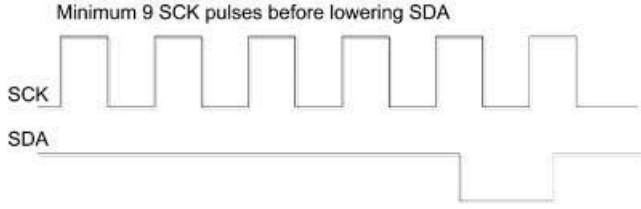


Figure 12.36 The SHT11 RESET sequence

www.it-ebooks.info

specifies that after the SCK changes state it must remain in its new state for a minimum of 100 ns. Here, a delay of 1 μs is introduced between each SCK state change:

```
void Reset_Sequence()
{
    SCK = 0;                    // SCK low
    SDA_Direction = 1;          // Define SDA as input so that the SDA line becomes HIGH
    for (j = 0; j < 10; j++)    // Repeat 10 times
    {
         SCK = 1;               // send 10 clocks on SCK line with 1us delay
         Delay_us(1);           // 1us delay
         SCK = 0;               // SCK is LOW
         Delay_us(1);           // 1us delay
    }
    Transmission_Start_Sequence();  // Send Transmission-start-sequence
}
```

Notice that when the direction of a port pin is set to 1 (i.e. when in input mode), the port pin presents itself as a logic HIGH.

12.7.2 Transmission-Start-Sequence

Before a temperature or relative humidity conversion command is sent to the SHT11, the transmission-start-sequence must be sent. This sequence consists of lowering the DATA line while SCK is HIGH, followed by a pulse on SCK and rising DATA again while SCK is still HIGH. Figure 12.37 shows the transmission-start-sequence.

The C code to implement the transmission-start-sequence is given below:

```
void Transmission_Start_Sequence()
{
    SDA_Direction = 1;          // Set SDA HIGH
    SCK = 1;                    // SCK HIGH
    Delay_us(1);                // 1 us delay
    SDA_Direction = 0;          // SDA as output
    SDA = 0;                    // Set SDA LOW
    Delay_us(1);                // 1us delay
    SCK = 0;                    // SCK LOW
    Delay_us(1);                // 1us delay
    SCK = 1;                    // SCK HIGH
    Delay_us(1);                // 1 us delay
    SDA_Direction = 1;          // Set SDA HIGH
    Delay_us(1);                // 1us delay
    SCK = 0;                    // SCK LOW
}
```

www.it-ebooks.info

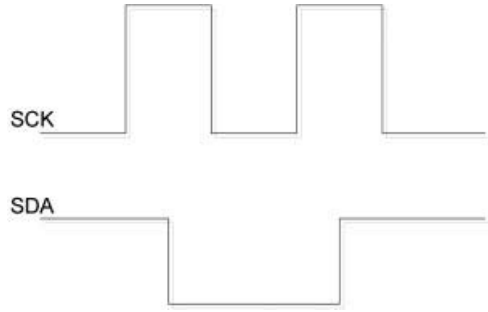


Figure 12.37 The transmission-start-sequence

12.7.3 Conversion Command

After sending the transmission-start-sequence, the device is ready to receive a conversion command. This consists of 3 address bits (only '000' is supported) followed by 5 command bits. The list of valid commands is shown in Table 12.1. For example, the commands for relative humidity and temperature are '00000101' and '00000011', respectively. After issuing a measurement command, the sensor sends an ACK pulse on the falling edge of the 8th SCK pulse. The ACK pulse is identified by the DATA line going LOW. The DATA line remains LOW until the 9th SCK pulse going LOW. The microcontroller then has to wait for the measurement to complete. This can take up to 320 ms. During this time, it is recommended to stop generating clocks on the SCK line and release the DATA line. When the measurement is complete, the sensor pulls the DATA line LOW to indicate that the data is ready. At this point, the microcontroller can restart the clock on the SCK line to read the measured data. Notice that the data is kept in the SHT11 internal memory until it is read out by the microcontroller.

The data readout consists of 2 bytes of data and 1 byte of CRC checksum. The checksum is optional and if not used the microcontroller may terminate the communication by keeping the DATA line HIGH after receiving the last bit of the data (LSB). The data bytes are transferred with MSB first and are right-justified. The measurement can be for 8, 12 or 14 bits wide. Thus, the 5th SCK corresponds to the MSB data for 12-bit operation. For 8-bit measurement, the first byte is not used. The microcontroller must acknowledge each byte by pulling the DATA line LOW, and sending a SCK pulse. The device returns to sleep mode after all the data has been read out.

Table 12.1 List of valid commands

Command	Code
00011	Measure temperature
00101	Measure relative humidity
00111	Read Status register
00110	Write Status register
11110	Soft Reset (reset interface, clear Status register)

www.it-ebooks.info

12.8 Acknowledgement

After receiving a command from the microcontroller, the sensor issues an acknowledgement pulse by pulling the DATA line LOW for one clock cycle. This takes place after the falling edge of the 8th clock on the SCK line, and the DATA line is pulled LOW until the end of the 9th clock on the SCK line.

12.8.1 The Status Register

The Status register is an internal 8 bit register that controls some functions of the device, such as selecting the measurement resolution, end of battery detection, and use of the internal heater. In order to write a byte to the Status register, the microcontroller must send the write command (‘00110’), followed by the data byte to be written. Note that the sensor generates acknowledge signals in response to receiving both the command and the data byte. Bit 0 of the Status register controls the resolution, such that when this bit is 1, both the temperature resolution and the relative humidity resolution are 12 bits. When this bit is 0, the temperature resolution is 14 bits and the relative humidity resolution is 12 bits.

The sensor includes an on-chip heating element that can be enabled by setting bit 2 of the Status register. By using the heater, it is possible to increase the sensor temperature by 5 to 10 °C. The heater can be useful for analysing the effects of changing the temperature on humidity. Notice that during temperature measurements, the sensor measures the temperature of the heated sensor element and not the ambient temperature.

The steps for reading the humidity and temperature are summarised below:

12.8.1.1 Humidity (assuming 12-bit operation with no CRC)

- Send Reset_Sequence.
- Send Transmission_start_sequence.
- Send ‘00000101’ to convert relative humidity.
- Receive ACK from sensor on 8th SCK pulse going LOW. The ACK is identified by the sensor lowering the DATA line.
- Wait for the measurement to be complete (up to 320 ms), or until DATA line is LOW.
- Ignore first 4 SCK pulses.
- Get the 4 upper nibble starting with the MSB bit.
- Send ACK to sensor at the end of 8th clock by lowering the DATA line and sending a pulse on SCK.
- Receive low 8 bits.
- Ignore CRC by keeping the DATA line HIGH.
- Next measurement can start by repeating the above steps.

12.8.1.2 Temperature

The steps for reading the temperature are similar, except that the command ‘00000011’ is sent instead of ‘00000101’

www.it-ebooks.info

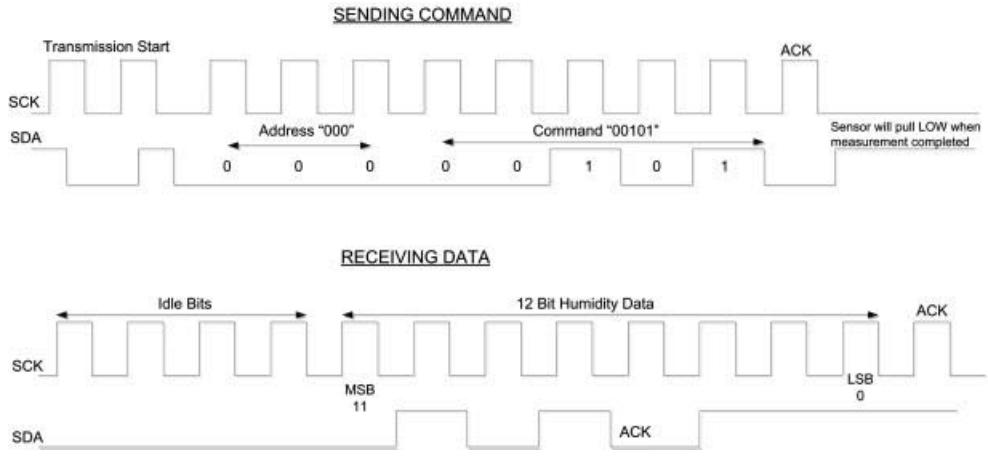


Figure 12.38 Reading the relative humidity with 12-bit resolution and no CRC

Figure 12.38 shows the timing diagram for reading the relative humidity with 12-bit resolution and ignoring the CRC. In this figure, it is assumed that the sensor has already been reset using the Reset_sequence.

12.8.2 Conversion of Signal Output

12.8.2.1 Relative Humidity Reading (SO_{RH})

The humidity sensor is non-linear and it is necessary to perform a calculation to obtain the correct reading. The manufacturer's data sheet recommends the following formula for the correction:

$$RH_{linear} = C_1 + C_2 + SO_{RH} + C_3.SO_{RH}^2(\%RH) \tag{12.7}$$

where SO_{RH} is the value read from the sensor and the coefficients are as given in Table 12.2.

For temperatures significantly different from the 25 °C, the manufacturers recommend another correction to be applied to the relative humidity as

$$RH_{TRUE} = (T - 25).(t_1 + t_2.SO_{RH}) + RH_{linear} \tag{12.8}$$

where T is the temperature in °C where the relative humidity reading is taken, and the coefficients are as given in Table 12.3.

Table 12.2 Coefficients for the RH non-linearity correction

SO_{RH}	C_1	C_2	C_3
12 bit	−2.0468	0.0367	−1.5955E-6
8 bit	−2.0468	0.5872	−4.0845E-4

www.it-ebooks.info

Table 12.3 Coefficients for RH temperature correction

SO_{RH}	t_1	t_2
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Table 12.4 Coefficients for temperature correction

VDD	d_1 (°C)	d_1 (°F)
5	−40.1	−40.2
4	−39.8	−39.6
3.5	−39.7	−39.5
3	−39.6	−39.3
2.5	−39.4	−38.9
SO_T	D_2(°C)	D_2(°F)
14 bit	0.01	0.018
12 bit	0.04	0.072

12.8.2.2 Temperature Reading (SO_T)

The manufacturers recommend that the temperature reading of the SHT11 should be corrected according to the following formula:

$$T_{TRUE} = d_1 + d_2.SO_T \tag{12.9}$$

where SO_T is the value read from the sensor and the coefficients are as given in Table 12.4.

12.8.3 Block Diagram

The block diagram of the project is shown in Figure 12.39. A rectangle will be drawn on the GLCD screen and the temperature and relative humidity will both be displayed, as shown in the figure.

12.8.4 Circuit Diagram

The circuit diagram of the project is as shown in Figure 12.40. The SHT11 sensor is connected to PORT C of a PIC18F45K22 type microcontroller. The DATA pin is connected to RC4 and the SCK pin is connected to RC3. The DATA pin is pulled up using a 10 KB resistor, as recommended by the manufacturers. Also, a 100 nF decoupling capacitor is connected between the VDD pin and the ground.

The project was tested using a plug-in SHT11 module (manufactured by mikroElektronika) together with the EasyPIC 7 development board, where the module was connected to the PORT C I/O plug located at the edge of the EasyPIC 7 development board (see Figure 12.41).

www.it-ebooks.info

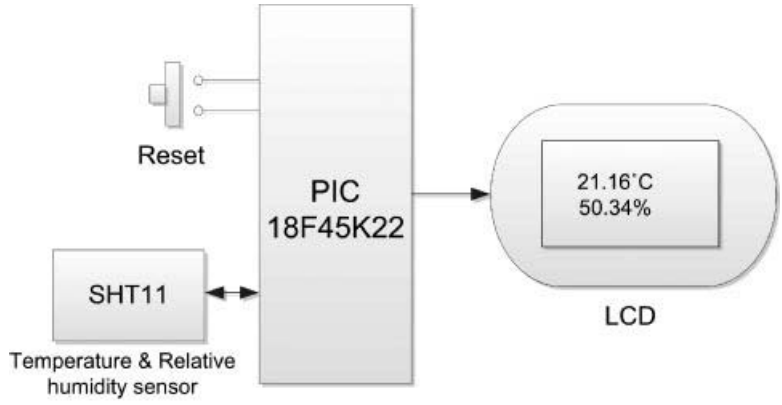


Figure 12.39 Block diagram of the project

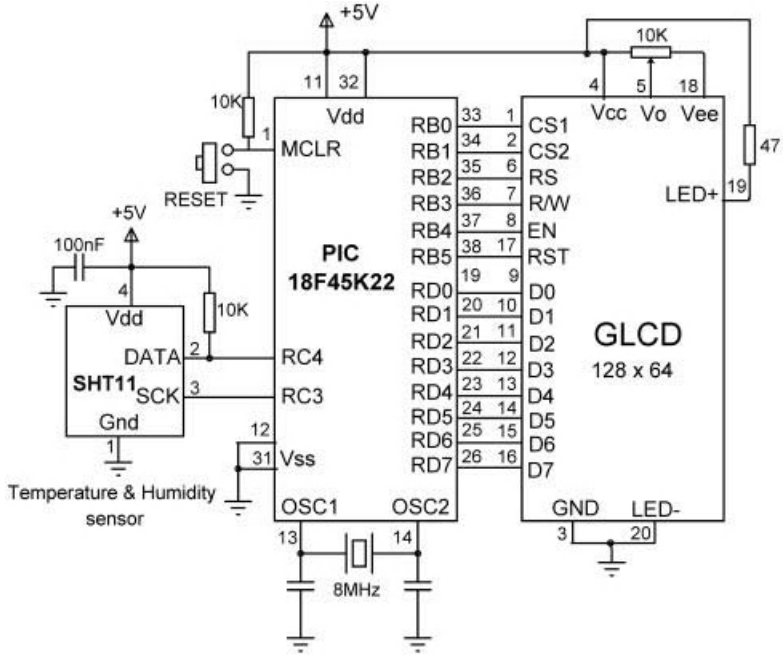


Figure 12.40 Circuit diagram of the project

www.it-ebooks.info

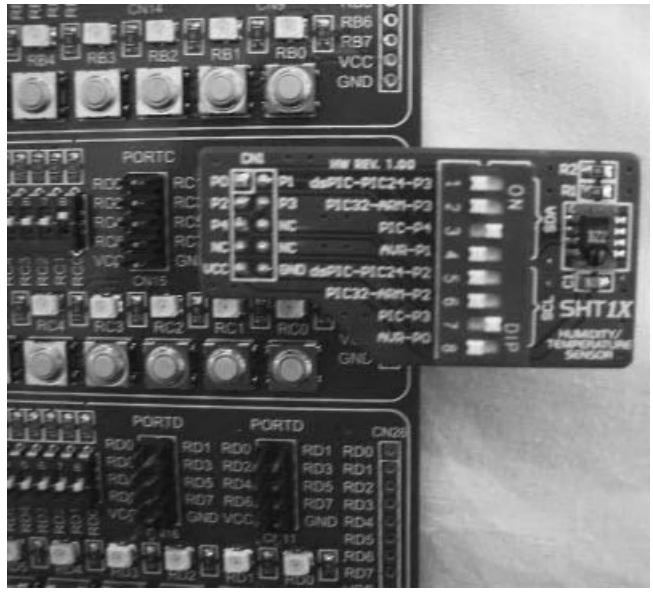


Figure 12.41 Using the SHT11 module with the EasyPIC 7 development board

12.8.5 Project PDL

The PDL of this project is given in Figure 12.42.

12.8.6 Project Program

The program listing of the project is shown in Figure 12.43. At the beginning of the program, the connections between the GLCD and the microcontroller are defined. Then, the connections between the SHT11 sensor and the microcontroller are defined. The temperature and relative humidity correction coefficients are then given as floating point numbers.

The main program then configures PORT B, PORT C and PORT D as digital outputs, initialises the GLCD, and clears the display. The GLCD font is set to large. The SHT11 datasheet recommends that no commands should be sent to the device for the first 20 ms after power is applied to the device. This delay is implemented by function SHT11_StartupDelay. The program then enters an endless loop formed with a *for* statement. Inside this loop, the temperature is measured and corrected by calling function Measure with argument 3, and stored in floating point variable T_{true}. Then, the relative humidity is read, corrected and stored in floating point variable RH_{true}. The measured values are converted into strings by using built-in function FloatToStr. Finally, the degree symbol and letter 'C' are appended to the temperature reading. Similarly, symbol '%' is appended to the relative humidity reading before it is displayed.

www.it-ebooks.info

```
BEGIN
    Define the connections between the GLCD and microcontroller
    Define the connections between the SHT11 and microcontroller
    Define SHT11 correction coefficients
    Configure PORTB, PORTC, PORTD as digital
    Initialise GLCD
    Clear display
    Set large fonts for GLCD
    CALL SHT11_Startup_Delay
    DO FOREVER
        CALL Measure to measure temperature
        CALL Measure to measure relative humidity
        Convert temperature to a string
        Convert relative humidity to a string
        Append degree symbol and letter "C" after the temperature value
        Append % sign after the relative humidity value
        Draw a rectangle on GLCD screen
        Display temperature string
        Display relative humidity string
        Wait for 5 seconds
    ENDDO
END

BEGIN/SHT11_Startup_Delay
    Wait for 20ms
END/SHT11_Startup_Delay

BEGIN/Reset_Sequence
    Implement SHT11 reset sequence
END/Reset_Seqeunce

BEGIN/Transmission_Start_Sequence
    Implement SHT11 transmission_start_sequence
END/Transmission_Start_Sequence

BEGIN/Send_ACK
    Send ACK signal to SHT11
END/Send_ACK

BEGIN/Measure
    Get type of measurement required
    Send Reset_Sequence
    Send Transmission_Start_Sequence
    Send address and temperature or humidity convert command to SHT11
    Send SCK pulse for the ACK signal
    Wait until measurement is ready (until DATA goes LOW)
    Read 8 bit measurement data
    Send ACK to SHT11
    Read remaining 8 bits
    Make corrections for temperature (or humidity)
END/Measure
```

Figure 12.42 PDL of the project

www.it-ebooks.info

```
/*=====================================================================
        TEMPERATURE AND RELATIVE HUMIDITY MEASUREMENT
        =============================================

This projects measures both the ambient temperature and the relative humidity and then
displays the readings on the GLCD.

The SHT11 single chip temperature and relative humidity sensor is used in this project. The
sensor is connected as follows to a PIC18F45K22 type microcontroller operating at 8 MHz:

 Sensor   Microcontroller Port
 DATA       RB0
 SCK        RB1

A 10K pull-up resistor is used on the DATA pin. In addition, a 100nF decoupling capacitor is
used between the VDD and the GND pins. The sensor is operated from a +5V supply.

The connections between the GLCD and the microcontroller is as in the earlier GLCD based
projects. The GLCD brightness is adjusted using a 10K potentiometer.


Author:    Dogan Ibrahim
File:      GLCD7.C
Date:      December 2011
=====================================================================*/

// GlCD module connections
char GLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbit GLCD_RS_Direction  at TRISB2_bit;
sbit GLCD_RW_Direction  at TRISB3_bit;
sbit GLCD_EN_Direction  at TRISB4_bit;
sbit GLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

//SHT11 connections
sbit SHT11_SDA at RC4_bit;                    // SHT11 DATA pin
sbit SHT11_SCK at RC3_bit;                    // SHT11 SCK pin
sbit SHT11_SDA_Direction at TRISC4_bit;       // DATA pin direction
sbit SHT11_SCK_Direction at TRISC3_bit;       // SCK pin direction

//
```

Figure 12.43 Program listing of the project

www.it-ebooks.info

```
// SHT11 Constants for calculating humidity (in 12 bit mode)
//
const float C1 = -2.0468;                          // -2.0468
const float C2 = 0.0367;                           // 0.0367
const float C3 = -1.5955E-6;                       // -1.5955* 10^-6

//
// SHT11 Constants for relative humidity temperature correction (in 12 bit mode)
//
const float t1 = 0.01;                             // 0.01
const float t2 = 0.00008;                          // 0.00008

//
// SHT11 temperature conversion coefficients (14 bit mode)
//
const float d1 = -40.1;                            // -40.1
const float d2 = 0.01;                             // 0.01

unsigned char i, mode;
unsigned int buffer;
float Res, Ttrue, RHtrue;
char T[14], H[14];

//
// Function to send the Transmission_Start_Sequence
//
void Transmission_Start_Sequence(void)
{
  SHT11_SDA_Direction = 1;                         // Set SDA as input
  SHT11_SCK = 1;                                   // SCK HIGH
  Delay_us(1);                                     // 1us delay
  SHT11_SDA_Direction = 0;                         // Set SDA as output
  SHT11_SDA = 0;                                   // SDA LOW
  Delay_us(1);                                     // 1us delay
  SHT11_SCK = 0;                                   // SCK LOW
  Delay_us(1);                                     // 1us delay
  SHT11_SCK = 1;                                   // SCK HIGH
  Delay_us(1);                                     // 1us delay
  SHT11_SDA_Direction = 1;                         // Set SDA as input
  Delay_us(1);                                     // 1us delay
  SHT11_SCK = 0;                                   // SCK low
}

//
// This function send the Reset_Sequence
//
void Reset_Sequence()
{
 SHT11_SCK = 0;                                    // SCL low
 SHT11_SDA_Direction = 1;                          // define SDA as input
 for (i = 1; i<= 10; i++)                          // repeat 10 times
```

Figure 12.43 (*Continued*)

www.it-ebooks.info

```
{
  SHT11_SCK = 1;                                  // Send clock pulses
  Delay_us(1);
  SHT11_SCK = 0;
  Delay_us(1);
}
Transmission_Start_Sequence();
}


//
// This function sends ACK
//
void Send_ACK()
{
  SHT11_SDA_Direction = 0;                        // define SDA as output
  SHT11_SDA = 0;                                  // SDA low
  SHT11_SCK = 1;                                  // SCL high
  Delay_us(1);                                    // 1us delay
  SHT11_SCK = 0;                                  // SCL low
  Delay_us(1);                                    // 1us delay
  SHT11_SDA_Direction = 1;                        // define SDA as input
}


//
// This function returns temperature or humidity depending on the argument
//
float Measure(unsigned char command)
{
  mode = command;                                 // mode is 3 or 5
  Reset_Sequence();                               // Reset SHT11
  Transmission_Start_Sequence();                  // Start transmission sequence transmission

   SHT11_SDA_Direction = 0;                       // Set SDA as output
   SHT11_SCK = 0;                                 // Set SCK as LOW
//
// Send address and command to SHT11 sensor. A total of 8 bits are sent
//
   for(i = 0; i< 8; i++)                          // Send address and command
   {
     if (mode.F7 == 1)SHT11_SDA_Direction = 1;    // if MSB (bit 7) is 1, Set SDA to 1
     else                                         // if MSB is 0
     {                                            // else MSB is 0
       SHT11_SDA_Direction = 0;                   // define SDA as output
       SHT11_SDA = 0;                             // Set SDA to 0
     }
     Delay_us(1);                                 // 1us delay
     SHT11_SCK = 1;                               // SCL high
     Delay_us(1);                                 // 1us delay
     SHT11_SCK = 0;                               // SCL low
```

Figure 12.43 (*Continued*)

www.it-ebooks.info

```
    mode = mode << 1;                          // move contents of j one place left
   }
//
// Give a SCK pulse for the ACK
//
  SHT11_SDA_Direction = 1;                     // Set SDA to input (to read ACK)
  SHT11_SCK = 1;                               // SCL high
  Delay_us(1);                                 // 1us delay
  SHT11_SCK = 0;                               // SCL low
  Delay_us(1);                                 // 1us delay
//
// Now wait until the measurement is ready (SDA goes LOW when data becomes ready)
//
  while (SHT11_SDA == 1)Delay_us(1);           // wit until SDA goes LOW
//
// Now, the data is ready, read the data as 2 bytes. Read all 16 bits even though the
// upper nibble may not be relevant
//
 buffer = 0;
 for (i = 1; i <=16; i++)                      // DO 16 times
 {
   buffer = buffer << 1;                       // move contents of MSB one place left
   SHT11_SCK = 1;                              // SCK HIGH
   if (SHT11_SDA == 1)buffer = buffer | 0x0001; // Get the bit as 1 (OR with existing data)
   SHT11_SCK = 0;
   if (i == 8)Send_ACK();                      // if counter i = 8 then send ACK
 }

//
// Now make the corrections to the measured value. If mode=3 then temperature, if on the
// other hand, mode=5 then relative humidity
//
 if(command == 0x03)                           // Temperature correction
   Res = d1 + d2*buffer;
 else if(command == 0x05)                      // Relative humidity correction
 {
   Res = C1 + C2*buffer + C3*buffer*buffer;
   Res = (Ttrue - 25)*(t1 + t2*buffer) + Res;
 }
 return Res;                                   // Return temperature or humidity
}


//
// This is the SHT11 startup delay (20ms)
//
void SHT11_Startup_Delay()
{
Delay_ms(20);
}
```

Figure 12.43 (*Continued*)

www.it-ebooks.info

```
//
// Start of MAIN program
//
void main()
{
  ANSELB = 0;                                          // Configure PORT B as digital
  ANSELC = 0;                                          // Configure PORT C as digital
  ANSELD = 0;                                          // Configure PORT D as digital
  TRISB = 0;
  TRISC = 0;
  SHT11_SCK_Direction = 0;                             // SCL is output

  GLCD_Init();                                         // initialiseGLCD
  Glcd_Fill(0);                                        // Clear GLCD
  Glcd_Set_Font(Font_Glcd_Character8x7, 8, 7, 32);     // Set large fonts
  SHT11_Startup_Delay();                               // SHT11 startup delay

  for(;;)                                              // DO FOREVER
  {
    SHT11_SCK_Direction = 0;                           // define SCL1 as output
    Ttrue = Measure(0x03);                             // Measure Temperature
    RHtrue = Measure(0x05);                            // Measure Relative humidity
    SHT11_SCK_Direction = 1;                           // define SCK as input
    FloatToStr(Ttrue, T);                              // Convert temperature to string
    FloatToStr(RHtrue, H);                             // Convert relative humidity to string

    Glcd_Fill(0);                                      // Clear display
    T[5] = 248;                                        // Insert Degree sign
    T[6] = 'C';                                        // Insert C
    T[7] = 0x0;                                        // Terminate with NULL
    H[5] = '%';                                        // Insert %
    H[6]=0x0;                                          // Terminate with NULL
    Glcd_Rectangle(20, 8, 100, 45, 1);                 // Draw a rectangle
    Glcd_Write_Text(T, 30, 2, 1);                      // Display temperature
    Glcd_Write_Text(H, 30, 4, 1);                      // Display Relative humidity
    Delay_ms(5000);                                    // Delay 5 seconds
  }
}
```

Figure 12.43 (*Continued*)

Function Measure is the most complicated function in the program. This function implements the measurement steps described earlier in the project. Argument *command* specifies the type of measurement required: 3 for temperature measurement and 5 for relative humidity measurement. After calling to functions Reset_Sequence and Transmission_Start_Sequence, the address and command is sent to the SHT11 device. Bits of *mode* (3 or 5) are sent out through the MSB after shifting the data to the left in a loop. The program then waits until the conversion is ready, which is indicated by the DATA line going LOW. Once the data is ready, a loop is formed to read the two bytes from the sensor. At the end of the 8th clock pulse, an ACK signal is sent to the sensor. In the last part of this function, depending upon the type of conversion required, either the temperature or the relative humidity readings are corrected and returned to the calling program.

The program repeats after a delay of 5 seconds.

www.it-ebooks.info

Figure 12.44 A typical display

Figure 12.44 shows a typical display.

12.9 Summary

This chapter has described the design of microcontroller based projects using GLCDs. At the beginning of the chapter, the creation and displaying of bitmap images are explained. It is shown how easy it is to create a bitmap image using the Windows Paint program. Then, a simple animation program is given, where a ball moves across the screen inside a rectangle shape. More complex projects, such as a GLCD dice, and the plotting of X-Y graphics, are given in later sections. The measurement and display of temperature is then given as a project, where the temperature is plotted on the GLCD screen. Finally, a project is given, where the ambient temperature and relative humidity are measured and displayed on the GLCD screen.

Exercises

12.1 Use the Windows Paint program to create an image with dimensions 128×64 pixels. Show how you can display this image on the GLCD.

12.2 Explain how a bitmap image of any size can be converted into 128×64 pixels and how it can be displayed on the GLCD.

12.3 Modify the moving ball animation program given in Figure 12.12, so that the ball moves diagonally.

12.4 Write a program to plot the function $y = 2x^2 + x - 12$, with x varying between -5 and $+5$.

12.5 Write a program to plot the function $y = 2x + 4$, with x varying between -1 and $+2$.

www.it-ebooks.info

13

Touch Screen Graphics LCD Projects

In this chapter we will look at the design of projects using graphics LCDs (GLCDs) with touch screen panels. As in Chapter 12, the GLCDs used in the projects in this section are 128 × 64 pixel monochrome, 78 × 70 × 14.3 mm displays, based on the KS108/KS107 controller. A 4-wire resistive touch screen (model no: TTW4028001) is used in the projects. Two projects are given in this chapter. The first project is simple and shows how the touch screen can be utilised to turn an LED ON and OFF. The second project is more complicated and shows how to flash an LED at a rate chosen by the user by touching the screen.

13.1 PROJECT 13.1 – Touch Screen LED ON-OFF

13.1.1 Project Description

This is perhaps the simplest GLCD project utilising touch screen. In this project, an LED is connected to pin RC7 of the microcontroller. Two buttons are placed on the screen, labelled ON and OFF. Touching the ON button will turn the LED ON. Similarly, touching the OFF button will turn the LED OFF.

The principle of operation of a touch screen is such that, as shown in Figure 13.1, if one side of a layer is connected to +V and the other side to ground, a potential gradient results on the screen layer, and the voltage at any point on this layer becomes directly proportional to the distance from the +V side.

In a 4-wire resistive touch screen, two measurements are made one after the other one to determine the X and Y co-ordinates of the point touched by the user. Figure 13.2a shows how the X co-ordinate can be determined. Here, the right- and left-hand sides of the top layer can be connected to +V and ground, respectively. The bottom layer can then be used to sense and measure the voltage at the point touched by the user. An A/D converter is used to convert this voltage to digital and then determine the X co-ordinate. Similarly, Figure 13.2b shows how

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

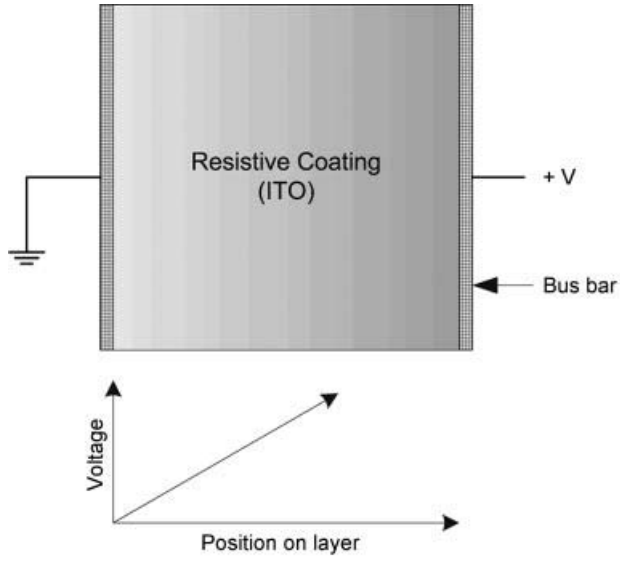


Figure 13.1 Voltage gradient in a screen layer

the Y co-ordinate can be determined. Here, the upper and lower sides of the bottom layer can be connected to +V and ground, respectively. The top layer can then be used to sense and measure the voltage at the point touched by the user. Again, an A/D is used to convert the voltage to digital and then to determine the Y co-ordinate.

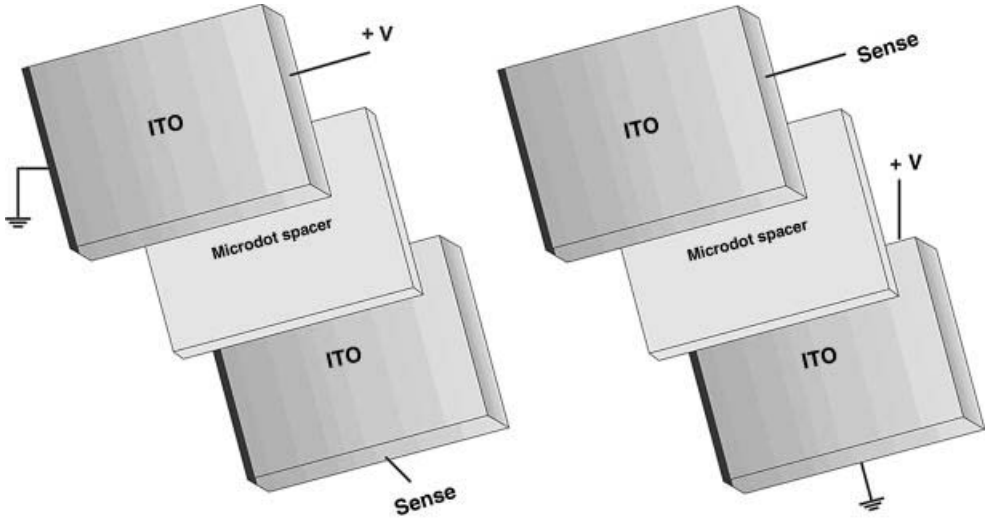


Figure 13.2 Determining the X and Y co-ordinates. (a) Determining the X co-ordinate (b) Determining the Y co-ordinate

www.it-ebooks.info

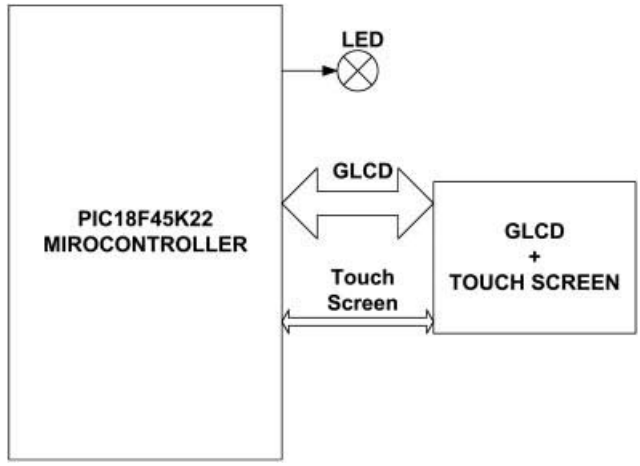


Figure 13.3 Block diagram of the project

13.1.2 Block Diagram

The block diagram of the project is shown in Figure 13.3. The touch screen graphics display will show the images, as shown in Figure 13.4. Rectangles and boxes will be drawn on the screen with text inside them. The screen is 128 pixels horizontal and 64 pixels vertical, with the origin at the top left corner, the X axis to the right and the Y axis downwards. The co-ordinates of the shapes are also shown in Figure 13.4.

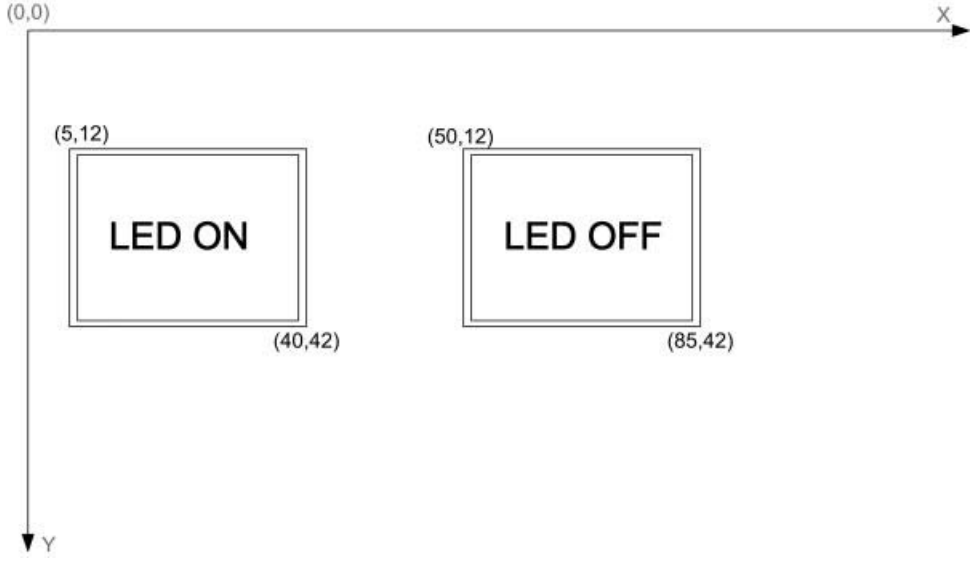


Figure 13.4 Screen layout

www.it-ebooks.info

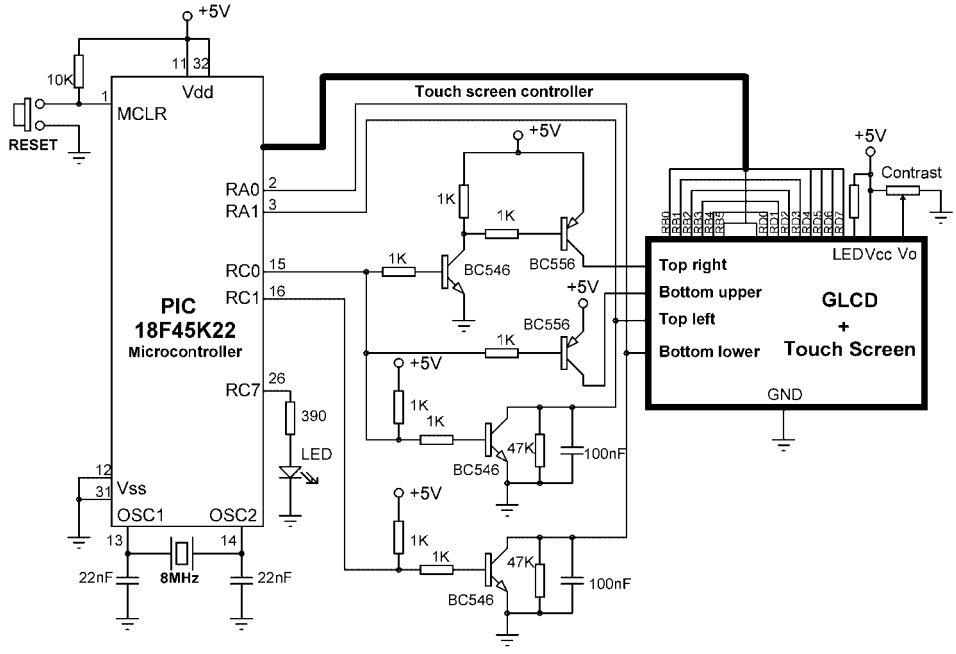


Figure 13.5 Circuit diagram of the project

The operation of the project is as follows: After power-up, the LED is OFF. The user can then touch the LED ON box to turn ON the LED. Touching the LED OFF box will turn the LED OFF.

13.1.3 Circuit Diagram

The circuit diagram of the project is shown in Figure 13.5. PORT B and PORT D of the microcontroller are connected to the GLCD. The connections between the microcontroller and the GLCD are as follows:

GLCD Pin	Microcontroller Pin
D0–D7	RD0-RD7
CS1	RB0
CS2	RB1
RS	RB2
R/W	RB3
E	RB4
RST	RB5

The background light of the GLCD is turned ON permanently by connecting the LDE input to +5 V via a resistor, and the GLCD contrast is adjusted using a 10 KB potentiometer.

In addition, as we shall see later in this section, the touch panel uses pins RA0, RA1, RC0 and RC1 of the microcontroller.

www.it-ebooks.info

In a microcontroller touch screen interface, a controller circuit is usually required to provide the correct logic levels to the touch screen pins. Normally, logic 0, logic 1 and OFF state are required. The OFF state can be provided using an open-drain microcontroller pin in input mode. Alternatively, touch screen controller chips, such as AD785 or AD7846, can be used to provide the necessary interface voltage levels. In circuit 13.5, switching transistors are used as the touch screen controller. For example, when RC0 is set to logic 1, Top Right pin becomes 1, Top Left pin becomes 0 and Bottom Upper pin becomes OFF.

13.1.3.1 Measuring the X Co-ordinate

In reference to Figure 13.6, and assuming the top layer has contacts Top Right and Top Left and the bottom layer has contacts Bottom Upper and Bottom Lower, the following setup is required to determine the X co-ordinate:

Top Left:	Ground
Top Right:	+5 V
Bottom Lower:	To A/D converter (X co-ordinate)
Bottom Upper:	OFF

Similarly, to determine the Y co-ordinate, the following setup should be made:

Top Left:	To A/D converter (Y co-ordinate)
Top Right:	OFF
Bottom Lower:	Ground
Bottom Upper:	+5 V

In Figure 13.6 for example, the X co-ordinate can be read into analogue port RA0 when:

RC0 = 1	(Top Left = 0, Top Right = 1, Bottom Upper = OFF)
RC1 = 0	(Bottom Lower = OFF)
Read RA0	(Read Bottom Lower)

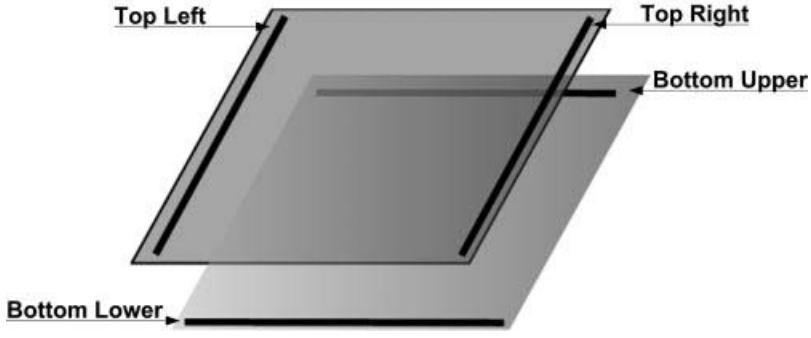


Figure 13.6 Connection of the touch screen

www.it-ebooks.info

Similarly, to read the Y co-ordinate:

RC0 = 0	(Top Left = OFF, Top Right = OFF, Bottom Upper = 1)
RC1 = 1	(Bottom Lower = Ground)
Read RA1	(Read Top Left)

If you are using the EasyPIC 7 development board, connect the touch screen flat cable, and then connect the GLCD into its socket. Set switch SW3 jumpers 4 to 7 to ON position to connect the RA0, RA1, RC0 and RC1 pins to the touch-screen controller circuit (see the EasyPIC 7 development board User Guide for further information).

13.1.4 Project PDL

The PDL of this project is very simple and is given in Figure 13.7.

```
BEGIN
    Configure PORT B, PORT C, and PORT D as digital output
    Configure PORT A as analog input
    Initialise GLCD
    Clear GLCD
    Draw Boxes for the two buttons
    Write texts inside the two boxes
    DO FOREVER
        CALLReadX to read X axis co-ordinate and convert to screen pixels
        CALLReadY to read Y axis co-ordinate and convert to screen pixels
        IF X and Y axis co-ordinates are inside the LED ON box THEN
            Turn ON the LED
        ENDIF
        IF X and Y axis co-ordinates are inside the LED OFF box THEN
            Turn OFF the LED
        ENDIF
    ENDDO
END


BEGIN/ReadX
    Set RC0 and RC1 to read the X axis co-ordinate
    Read Channel 0 (RA0 or AN0)
    Convert to X axis pixel co-ordinates
    Return pixel co-ordinates
END/ReadX


BEGIN/ReadY
    Set RC0 and RC1 to read the Y axis co-ordinate
    Read Channel 1 (RA1 or AN1)
    Convert to Y axis pixel co-ordinates
    Return pixel co-ordinates
END/ReadY
```

Figure 13.7 PDL of the project

www.it-ebooks.info

13.1.5 Project Program

The program is named TScreen1.C and the program listing of the project is shown in Figure 13.8. At the beginning of the project, symbols ON and OFF are defined as 1 and 0,

```
/*************************************************************************************
                        GLCD TOUCH SCREEN LED
                        =====================

This project shows how the touch screen can be used with the GLCD. In this project, an LED
LED is connected to port RC7 of the microcontroller. Two buttons labelled LED ON and LED
OFF are placed on the GLCD screen. Touching the LED ON button turns on the LCD. Similarly,
touching the LED OFF button turns off the LCD.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin      Microcontroller pin
   CS1           RB0
   CS2           RB1
   RS            RB2
   R/W           RB3
   RST           RB4
   EN            RB5
   D0 – D7       RD0 – RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    TScreen1.C
*************************************************************************************/
#define ON 1
#define OFF 0
sbit LED at RC7_bit;

// Glcd module connections
charGLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbitGLCD_RS_Direction  at TRISB2_bit;
sbitGLCD_RW_Direction  at TRISB3_bit;
```

Figure 13.8 Program listing of the project

www.it-ebooks.info

```
sbitGLCD_EN_Direction  at TRISB4_bit;
sbitGLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

//
// This function reads the voltage of the point touched by the user. The voltage is read from
// Channel 0 (RA0). The voltage is then converted into the real pixel co-ordinates by multiplying
// it with 128/1024 (10-bit A/D converter has 1024 steps, and the screen has 128 pixels in the
// X direction)
//
longReadX(void)
{
  long x;
  RC0_bit = 1;                                  // Set to read X co-ordinate
  RC1_bit = 0;                                  //
  Delay_Ms(5);
  x = ADC_Read(0);                              // Read from Channel 0
  x = x*128/1024;                               // Convert to screen pixel co-ordinates
  return(x);
}


//
// This function reads the voltage of the point touched by the user. The voltage is read from
// Channel 1 (RA1). The voltage is then converted into the real pixel co-ordinates by multiplying
// with 64/1024 and taking away from 64 (the Y co-ordinate is downwards and it has 64 pixels)
//
longReadY(void)
{
  long y;
  RC0_bit = 0;                                  // Set to read Y co-ordinate
  RC1_bit = 1;                                  //
  Delay_Ms(5);
  y = ADC_Read(1);                              // Read from Channel 1
  y = 64 - ((y*64)/1024);                       // Convert to screen pixel co-ordinates
  return(y);
}


//
// Start of main program
//
void main()
{
  longx_real, y_real;
   ANSELA = 3;                                  // Configure RA0, RA1 as analog
   ANSELB = 0;                                  // Configure PORT B as digital
   ANSELC = 0;                                  // Configure PORT C as digital
   ANSELD = 0;                                  // Configure PORT D as digital
   TRISA = 0x03;                                // RA0 and RA1 are inputs
   TRISB = 0;                                   // PORT B is output
```

Figure 13.8 (*Continued*)

www.it-ebooks.info

```
  TRISD = 0;                                          // PORT D is output
  TRISC = 0;                                          // PORT C is output
  PORTA = 0;

  Glcd_Init();                                        // Initialise GLCD
  Glcd_Fill(0x0);                                     // Clear GLCD
  LED = OFF;                                          // Turn OFF the LED to start with

  Glcd_Rectangle(5, 12, 40, 42, 1);                   // Draw LED ON rectangle
  Glcd_Box(7, 14, 38, 40, 1);                         // Draw LED ON box
  Glcd_Rectangle(50, 12, 85, 42, 1);                  // Draw LED OFF rectangle
  Glcd_Box(52, 14, 83, 40, 1);                        // Draw LED OFF box

  Glcd_Set_Font(Font_Glcd_System3x5,3,5,32);          // Change font to smaller size
  Glcd_Write_Text("LED ON", 10, 3, 0);                // Write LED ON
  Glcd_Write_Text("LED OFF", 54, 3, 0);               // Write LED OFF

  for(;;)                                             // Do FOREVER
  {
    x_real = ReadX();                                 // Read X co-ordinate
    y_real = ReadY();                                 // Read Y co-ordinate
    if((x_real>= 7 &&x_real<= 38) && (y_real>= 14 &&y_real<=40))LED = ON;
    if((x_real>= 52 &&x_real<= 83) && (y_real>= 14 &&y_real<=40))LED = OFF;
  }
}
```

Figure 13.8 (*Continued*)

respectively, and LED is assigned to port pin RC7. Then, the connection between the microcontroller and the GLCD are defined using *sbit* statements. PORT B and PORT D are used to drive the GLCD and both of these ports are configured as digital output. PORT C is also configured as digital output, since the LED is connected to pin RC7 of this port. PORT A is configured as analogue and bits 0 and 1 of this port are configured as inputs, so that RA0 (or AN0) and RA1 (or AN1) become analogue input ports.

The GLCD is then initialised, the screen is cleared, and the LED is turned OFF at the beginning of the program. The program then draws two boxes with rectangle edges and writes the texts LED ON and LED OFF inside these boxes. The rectangles and boxes are drawn using the Glcd_Rectangle and Glcd_Box functions, respectively, at the following co-ordinates:

```
Glcd_Rectangle(5, 12, 40, 42, 1);                 // Draw LED ON rectangle
Glcd_Box(7, 14, 38, 40, 1);                       // Draw LED ON box
Glcd_Rectangle(50, 12, 85, 42, 1);                // Draw LED OFF rectangle
Glcd_Box(52, 14, 83, 40, 1);                      // Draw LED OFF box
```

The text font is changed to 3×5 and texts are written inside the boxes using the following GLCD functions:

```
Glcd_Set_Font(Font_Glcd_System3x5,3,5,32);
Glcd_Write_Text("LED ON", 10, 3, 0);
Glcd_Write_Text("LED OFF", 54, 3, 0);
```

www.it-ebooks.info

Figure 13.9 Typical display on the GLCD

The main part of the program is executed in an endless loop, formed using a *for* statement. Inside this loop, the X and Y co-ordinates of the screen are read by calling functions ReadX and ReadY. Function ReadX reads the analogue voltage corresponding to the point touched by the user, and returns the real X co-ordinate of this point as the pixel co-ordinate. Notice that since a 10-bit A/D converter is used, the voltage read is converted to the pixel co-ordinate by multiplying it with 128/1024. Similarly, function ReadY reads the analogue voltage corresponding to the point touched by the user and returns the real Y co-ordinate touched by the user as the pixel co-ordinate. Here also, the voltage read is converted to the pixel co-ordinate by multiplying it with 64/1024 and subtracting from 64, since the Y co-ordinate is downwards. Notice that the Y co-ordinate of the touch screen is not very accurate since the number of pixels in the Y direction is small and the voltage read is multiplied by 64/1024 = 0.0625. Thus, for example, if the converted digital value is 0 or 10, or even 15, the same value will be returned by the function. The program then checks to see whether or not the touched point is inside the shape labelled LED ON, and if so, the LED is turned ON. Similarly, the LED is turned OFF if the point touched by the user is inside the shape labelled LED OFF.

Figure 13.9 shows a typical display on the GLCD.

13.2 PROJECT 13.2 – LED Flashing with Variable Rate

13.2.1 Project Description

This project shows how the GLCD and touch screen can be used to develop the project of a flashing LED, where the flashing rate is selected by touching the appropriate button on the GLCDscreen. Figure 13.10 shows the screen layout and co-ordinates of the various boxes and rectangles displayed on the screen.

www.it-ebooks.info

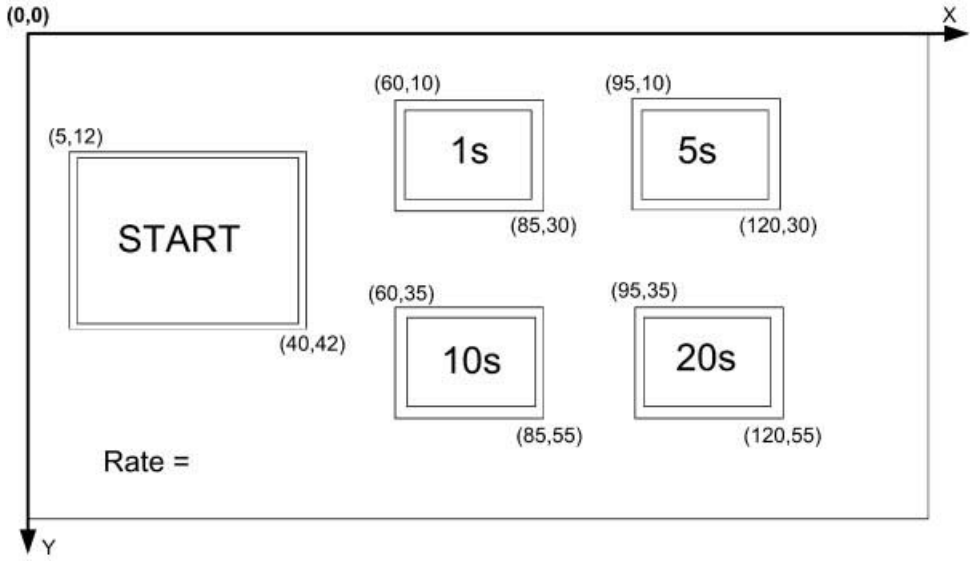


Figure 13.10 Screen layout of the project

An LED is connected to pin RC7 of the microcontroller. The flashing rate is displayed on the screen in the form of boxes. Four options are available: 1 second, 5 seconds, 10 seconds and 20 seconds. The user initially selects the flashing rate by touching the required box on the screen. The flashing then starts by touching the START button.

13.2.2 Block Diagram

The block diagram of the project is as shown in Figure 13.3.

13.2.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 13.5.

13.2.4 Project PDL

The PDL of this project is given in Figure 13.11.

13.2.5 Project Program

The program is named TScreen2.C and the program listing of the project is shown in Figure 13.12. At the beginning of the project, an LED is assigned to port pin RC7 of the

www.it-ebooks.info

```
BEGIN
    Define connections between GLCD and microcontroller
    Configure PORT B, PORT C and PORT D as digital outputs
    Define texts to be displayed inside the shapes
    Configure PORT A as analog input
    Initialise GLCD
    Clear GLCD
    CALL Background to display the background
    DO FOREVER
        CALLReadX to find the X co-ordinate of touched point
        CALLReadY to find the Y co-ordinate of touched point
        Find out which button is touched to determine the selected flashing rate
        Display the selected flashing rate
        IF START button is touched
            Display "flashing…" at the bottom of the screen
            DO FOREVER
                Flash the LED at the selected rate
            ENDDO
        ENDIF
    ENDDO
END


BEGIN/ReadX
    Setup to read the X co-ordinate of the touched point
    Read and convert the touched point to screen co-ordinates
END/ReadX


BEGIN/ReadY
    Setup to read the Y co-ordinate of the touched point
    Read and convert the touched point to screen co-ordinates
END/ReadY


BEGIN/Background
    Display rectangles at specified co-ordinates
    Display boxes at specified co-ordinates
    Display texts inside the shapes
END/Background
```

Figure 13.11 PDL of the project

microcontroller. Then, the connections between the microcontroller and the GLCD are defined using *sbit* statements. PORT B and PORT D are used to drive the GLCD and both of these ports are configured as digital outputs. PORT C is also configured as digital output, since the LED is connected to pin RC7 of this port. PORT A is configured as analogue and bits 0 and 1 of this port are configured as inputs so that RA0 (or AN0) and RA1 (or AN1) become analogue input ports.

The GLCD is then initialised, the screen is cleared, and the LED is turned OFF at the beginning of the program. The program then draws the boxes and rectangles, and writes the texts START, and the flashing rates inside these shapes. These boxes and rectangles are

www.it-ebooks.info

```
/*************************************************************************************************
                VARIABLE FLASHING LED WITH TOUCH SCREEN
                =======================================

This is another project showing how the touch screen can be used with the GLCD. In this
project, an LED is connected to port RC7 of the microcontroller. The LED flashes where the
flashing rate is selected by touching the required button on the GLCD screen. The user can
select 1s, 5s, 10s, and 20s as the flashing rate. Touching the START button starts the flashing.

In this project a KS0107/108 controller based GLCD with 128 x 64 pixels is connected to a
PIC18F45K22 type microcontroller, operated from an 8MHz crystal (any other type PIC
microcontroller can also be used if desired).

The GLCD is connected to PORT B of the microcontroller as follows:

 GLCD pin      Microcontroller pin
   CS1           RB0
   CS2           RB1
   RS            RB2
   R/W           RB3
   RST           RB4
   EN            RB5
   D0 – D7       RD0 – RD7

The brightness of the GLCD is controlled by connecting the arm of a 10K potentiometer to
pin Vo of the GLCD. The other arms of the potentiometer are connected to pin Vee and
+5V supply.


Author:  Dogan Ibrahim
Date:    December, 2011
File:    TScreen2.C
*************************************************************************************************/
#define ON 1
#define OFF 0
sbit LED at RC7_bit;                              // LED is connected to pin RC7

// Glcd module connections
charGLCD_DataPort at PORTD;

sbit GLCD_CS1 at RB0_bit;
sbit GLCD_CS2 at RB1_bit;
sbit GLCD_RS  at RB2_bit;
sbit GLCD_RW  at RB3_bit;
sbit GLCD_EN  at RB4_bit;
sbit GLCD_RST at RB5_bit;

sbit GLCD_CS1_Direction at TRISB0_bit;
sbit GLCD_CS2_Direction at TRISB1_bit;
sbitGLCD_RS_Direction  at TRISB2_bit;
sbitGLCD_RW_Direction  at TRISB3_bit;
```

Figure 13.12 Program listing of the project

www.it-ebooks.info

```
sbitGLCD_EN_Direction  at TRISB4_bit;
sbitGLCD_RST_Direction at TRISB5_bit;
// End Glcd module connections

charmsg_Start[] = "START";
char msg_1s[] = "1s";
char msg_5s[] = "5s";
char msg_10s[] = "10s";
char msg_20s[] = "20s";
charmsg_Rate[] = "Rate=               ";
charmsg_Flashing[] = "flashing...";

//
// This function introduces variable delay to the program. The delay in seconds is passed as
// an argument to the function
//
voidDelay_Seconds(char d)
{
  char i;
  for(i=0; i<d; i++)Delay_Ms(1000);
}


//
// This function reads the voltage of the point touched by the user. The voltage is read from
// Channel 0 (RA0). The voltage is then converted into the real pixel co-ordinates by
// multiplying it with 128/1024 (10-bit A/D converter has 1024 steps, and the screen has 128
// pixels in the X direction)
//
longReadX(void)
{
  long x;
  RC0_bit = 1;
  RC1_bit = 0;
  Delay_Ms(5);
  x = ADC_Read(0);                                    // Read from Channel 0
  x = x*128/1024;                                     // Convert to real co-ordinates
  return(x);
}

//
// This function reads the voltage of the point touched by the user. The voltage is read from
// Channel 1 (RA1). The voltage is then converted into the real pixel co-ordinates by
// multiplying with 64/1024 and taking away from 64 (the Y co-ordinate is downwards and
// it has 64 pixels)
//
longReadY(void)
{
  long y;
  RC0_bit = 0;
  RC1_bit = 1;
```

Figure 13.12 (*Continued*)

www.it-ebooks.info

```
  Delay_Ms(5);
  y = ADC_Read(1);                          // Read from Channel 1
  y = 64 - ((y*64)/1024);                   // Convert to real co-ordinates
  return(y);
}


//
// This function displays the GLCD Background. First, rectangles and boxes are drawn to
//representthe user selections. Then, texts are written inside these shapes
//
void Background(void)
{
  Glcd_Rectangle(5,12,40,42,1);             // START rectangle
  Glcd_Box(7,14,38,40,1);                   // START box
  Glcd_Rectangle(60,10,85,30,1);            // 1s rectangle
  Glcd_Box(62,12,83,28,1);                  // 1s box
  Glcd_Rectangle(95,10,120,30,1);           // 5s rectangle
  Glcd_Box(97,12,118,28,1);                 // 5s box
  Glcd_Rectangle(60,35,85,55,1);            // 10s rectangle
  Glcd_Box(62,37,83,53,1);                  // 10s box
  Glcd_Rectangle(95,35,120,55,1);           // 20s rectangle
  Glcd_Box(97,37,118,53,1);                 // 20s box

  Glcd_Write_Text(msg_Start,8,3,0);         // START text
  Glcd_Write_Text(msg_1s,67,2,0);           // 1s text
  Glcd_Write_Text(msg_5s,102,2,0);          // 5s text
  Glcd_Write_Text(msg_10s,65,5,0);          // 10s text
  Glcd_Write_Text(msg_20s,100,5,0);         // 20s text
  Glcd_Write_Text(msg_Rate,10,7,1);         // RATE= text
}


//
// Start of main program
//
void main()
{
  longx_real, y_real;
  char rate, i;

  ANSELA = 3;                               // Configure RA0, RA1 asanalog
  ANSELB = 0;                               // Configure PORT B as digital
  ANSELC = 0;                               // Configure PORT C as digital
  ANSELD = 0;                               // Configure PORT D as digital
  TRISA = 0x03;                             // RA0 and RA1 are inputs
  TRISB = 0;                                // PORT B is output
  TRISD = 0;                                // PORT D is output
  TRISC = 0;                                // PORT C is output
  PORTA = 0;
```

Figure 13.12 (*Continued*)

www.it-ebooks.info

```
  Glcd_Init();                                  // Initialise GLCD
  Glcd_Fill(0x0);                               // Clear GLCD
  LED = OFF;                                    // Turn OFF LED to start with
  Background();                                 // Display GLCD background

  for(;;)                                       // DO FOREVER
  {
    x_real = ReadX();                           // Read X co-ordinate
    y_real = ReadY();                           // Read Y co-ordinate
//
// Check to see which button (if any) is touched to
//
  if((x_real>= 62 &&x_real<= 83) && (y_real>= 12 &&y_real<= 28))
  {
      rate = 1; msg_Rate[5]='1'; msg_Rate[6]='s'; msg_Rate[7]=' ';
  }
  else if((x_real>= 97 &&x_real<= 118) && (y_real>= 12 &&y_real<= 28))
  {
      rate = 5; msg_Rate[5] = '5'; msg_Rate[6] = 's'; msg_Rate[7]=' ';
  }
  else if((x_real>= 62 &&x_real<= 83) && (y_real>= 37 &&y_real<= 53))
  {
      rate = 10; msg_Rate[5]='1'; msg_Rate[6]='0'; msg_Rate[7]='s';
  }
  else if ((x_real>= 97 &&x_real<= 118) && (y_real>= 37 &&y_real<= 53))
  {
      rate = 20; msg_Rate[5]='2'; msg_Rate[6]='0'; msg_Rate[7]='s';
  }
//
// Display the selected rate
//
  Glcd_Write_Text(msg_Rate,1,7,1);
//
// Check if START button is touched and if so start the flashing action
//
  if ((x_real>= 7 &&x_real<= 38) && (y_real>= 14 &&y_real<= 40))
  {
        for(i=0; i<11; i++)msg_Rate[i+9] = msg_Flashing[i];
        Glcd_Write_Text(msg_Rate,1,7,1);

        for(;;)                                 // Start of the flashing action
        {
          LED = ON;                             // Turn LED ON
          Delay_Seconds(rate);                  // Wait for the specified delay
          LED = OFF;                            // Turn LED OFF
          Delay_Seconds(rate);                  // Wait for the specified delay
        }
  }
 }
}
```

Figure 13.12 (*Continued*)

www.it-ebooks.info

drawn using the Glcd_Rectangle and Glcd_Box functions, respectively, at the following co-ordinates:

```
Glcd_Rectangle(5,12,40,42,1);          // START rectangle
Glcd_Box(7,14,38,40,1);                // START box
Glcd_Rectangle(60,10,85,30,1);         // 1s rectangle
Glcd_Box(62,12,83,28,1);               // 1s box
Glcd_Rectangle(95,10,120,30,1);        // 5s rectangle
Glcd_Box(97,12,118,28,1);              // 5s box
Glcd_Rectangle(60,35,85,55,1);         // 10s rectangle
Glcd_Box(62,37,83,53,1);               // 10s box
Glcd_Rectangle(95,35,120,55,1);        // 20s rectangle
Glcd_Box(97,37,118,53,1);              // 20s box
```

The texts are written inside these boxes at the following co-ordinates:

```
Glcd_Write_Text(msg_Start,8,3,0);      // START text
Glcd_Write_Text(msg_1s,67,2,0);        // 1s text
Glcd_Write_Text(msg_5s,102,2,0);       // 5s text
Glcd_Write_Text(msg_10s,65,5,0);       // 10s text
Glcd_Write_Text(msg_20s,100,5,0);      // 20s text
Glcd_Write_Text(msg_Rate,10,7,1);      // RATE= text
```

The program then checks to see if the user touched a rate selection box, and if so, the required flashing rate is selected and loaded into variable *rate*. When the user touches the START box, the flashing action begins where the LED is flashed continuously at the specified rate inside a *for* loop, and the text 'flashing . . . ' is displayed at the bottom of the screen.

Figure 13.13 shows a typical display on the GLCD before the flashing is started. In Figure 13.14 the display is shown after the user selected the 5-second flashing rate and started the flashing action.

Figure 13.13 Display before starting the flashing action

www.it-ebooks.info

Figure 13.14 Display after starting the flashing action

13.3 Summary

Touch screen GLCDs are used in many consumer and industrial control applications. Most electronic games, mobile phones, MP3 players, GPS systems, and so on all use the touch screen technology. This chapter has explained the use of touch panel screens in microcontroller based projects. Two tested and working projects are given in the chapter. The first project is simple and shows how to control an LCD using two shapes on a touch screen LCD. The second project shows how to change the flashing rate of an LED using touch screen GLCD, where the flashing rate is selected by touching appropriate boxes on the screen.

Exercises

13.1 4 LEDs are connected to upper nibble of PORT C. Design a touch screen based project with 4 boxes on the screen to represent each LED. Initially, all the LEDs should be OFF. Touching the box representing an LED should toggle the state of the corresponding LED.

13.2 Design an integer calculator using a touch screen GLCD. Your calculator should have the numeric keys 0 to 9, the basic four mathematical keys '$+ - {}^{*}/$', and the ENTER key. A display should be provided to see the results of calculations.

13.3 Design a calculator to convert from °C to °F. Your calculator should have keys 0 to 9, and an ENTER key. A display should also be provided to show the results of conversions.

13.4 Design a touch screen based project for drawing rectangular shapes on the screen. The user should touch the top left and bottom right co-ordinates of the rectangle to be drawn, and the program should draw the required rectangle.

www.it-ebooks.info

14

Using the Visual GLCD Software in GLCD Projects

Visual GLCD is a standalone software package used for the rapid development of graphical user interfaces for various types of GLCDs in embedded devices. With the help of the software, users can create screens and place drag-and-drop components on these screens, which can be used for building complex graphical applications. The Visual GLCD software generates code compatible with the mikroC Pro for PICcompiler and similar compilers developed by mikroElektronika.

Current version (2.50) of Visual GLCD supports the following features:

- 15 graphical user interface components;
- automatic code generation for compilers;
- multiple font support, such as regular, bold, italic, underline and strikeout;
- support for external memory;
- 'On Press/release' procedures when objects are clicked (event driven design);
- changing properties of multiple objects at the same time;
- zoom in/out option for each screen generated;
- show/hide grid options;
- print and print preview of current screen.

The Visual GLCD software must be installed before it can be used. Download the software from the mikroElektronika Web site http://www.visualglcd.com. Start up the software by double clicking on the appropriate icon. The Visual GLCD IDE will appear on the screen and you are now ready to use the software (you may need to have a licence or a dongle to use the software in full capacity).

The steps to create a graphics application using the Visual GLCD software are given below:

1. Create project files.
2. Configure project.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

3. Add screen and give it a name.
4. Place components on the screen and configure component properties.
5. Assign actions (or events) to components.
6. Generate the code.
7. Compile the code and load to the target microcontroller.

Several examples are given in this chapter, to show how the Visual GLCD software can be used in GLCD projects. The first project gives all the steps in detail, while the other projects simply describe the important points in each project.

14.1 PROJECT 14.1 – Toggle LED

14.1.1 Project Description

This is perhaps the simplest Visual GLCD based project. In this project, a standard GLCD with touch screen is used, as described in Chapter 13. An LED is connected to pin RC7 of the microcontroller and the LED is toggled when a shape is touched on the screen.

14.1.2 Block Diagram

The block diagram of the project is as shown in Figure 13.3.

The operation of the project is as follows: After power-up and screen calibration, the user can touch the TOGGLE LED box to turn ON the LED. Touching this box again will turn the LED OFF.

14.1.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 13.5. Standard 128 × 64 pixel GLCD is used in this project. Switching transistors are used as the touch screen controller. A PIC18F45K22 type microcontroller is used in the design with 8 MHz crystal.

1. *Create Project File*
   Start the Visual GLCD software. Select Project -> New from the top-down menu. A window will appear, as shown in Figure 14.1. Enter the project name and choose the desired project path. Then click OK. In this project, the name TOGGLE is given to the project.
2. *Configure Project*
   In the Project Settings window, make the following settings:

14.1.3.1 General (see Figure 14.2)

- *Hardware patterns*: EasyPIC6;
- *Target Compiler*: mikroC Pro for PIC PRO for PIC;
- *Target Device*: PIC18F45K22;
- *Device Clock (Hz)*: 8 000 000;

www.it-ebooks.info

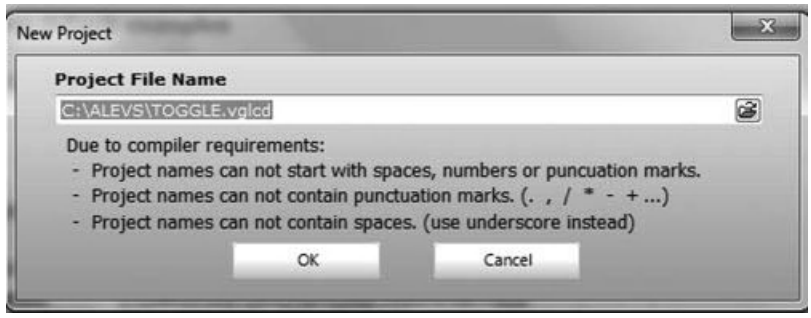

New Project
Project File Name
C:\ALEVS\TOGGLE.vglcd
Due to compiler requirements:
- Project names can not start with spaces, numbers or puncuation marks.
- Project names can not contain punctuation marks. (. , / * - +...)
- Project names can not contain spaces. (use underscore instead)
OK
Cancel

Figure 14.1 Specify the project name and project path

- Configure PORT B, PORT C and PORT D as digital I/O, using the appropriate statements for the chosen microcontroller (e.g. use the ANSEL statements if using the PIC18F45K22 microcontroller), and configure PORT C as output in the 'Init Code:' section.

Notice here that the EasyPIC6 development board is used in the project, as this board uses the standard GLCD and touch screen to microcontroller interface.

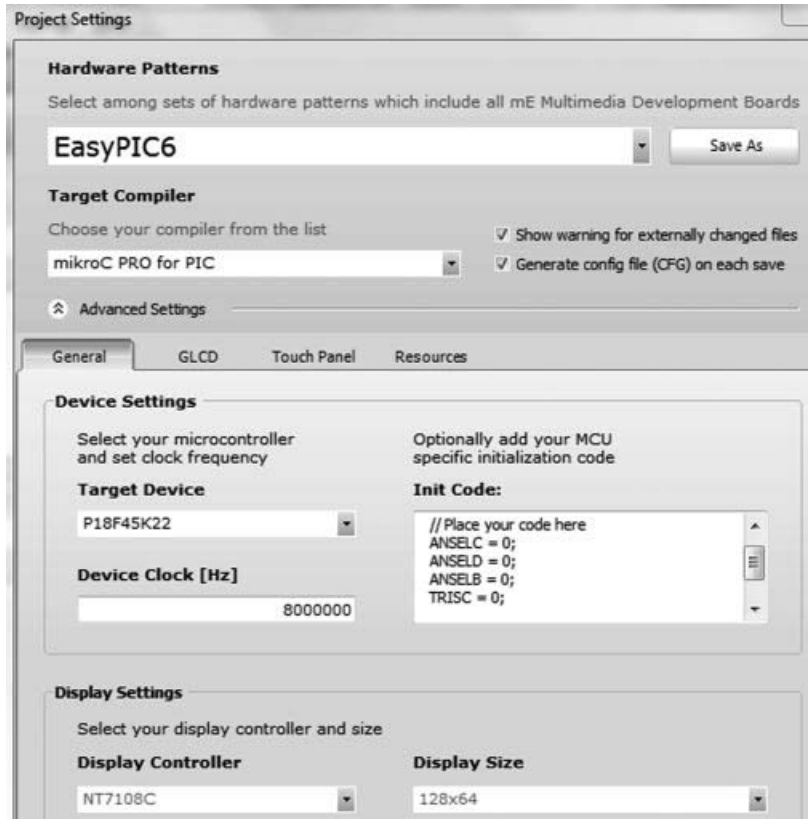


Figure 14.2 General settings

www.it-ebooks.info

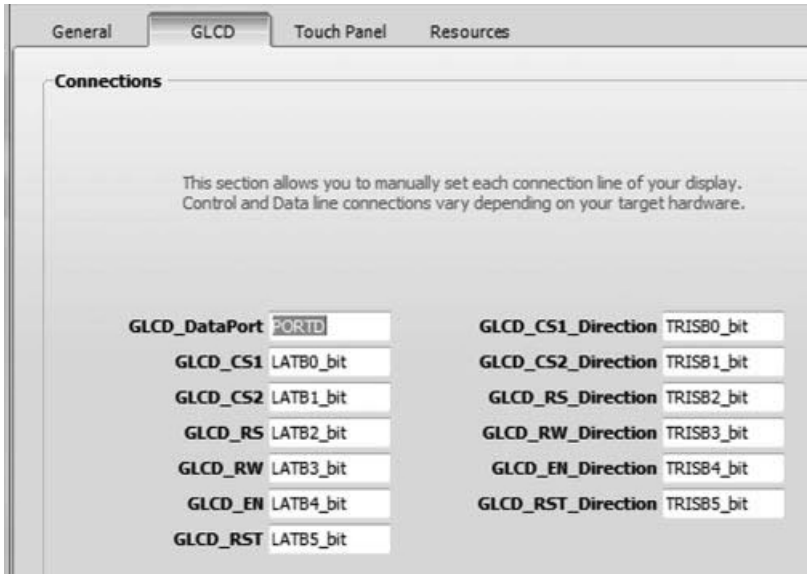


Figure 14.3 GLCD settings

14.1.3.2 GLCD (see Figure 14.3)

• GLCD_Data_Port:	PORTD	GLCD_CS1_Direction: TRISB0_bit
• GLCD_CS1:	LATB0_bit	GLCD_CS2_Direction: TRISB1_bit
• GLCD_CS2:	LATB1_bit	GLCD_RS_Direction: TRISB2_bit
• GLCD_RS:	LATB2_bit	GLCD_RW_Direction: TRISB3_bit
• GLCD_RW:	LATB3_bit	GLCD_EN_Direction: TRISB4_bit
• GLCD_EN:	LATB4_bit	GLCD_RST_Directiion:TRISB5_bit
• GLCD_RST:	LATB5_bit	

14.1.3.3 Touch Panel (see Figure 14.4)

```
READ-X: A/D Channel: 0    Drive A at: RC0_bit    DriveA_Direction at: TRISC0_bit
READ-Y: A/D Channel: 1    Drive B at: RC1_bit    DriveB_Direction at: TRISC1_bit
```

You should also check the section 'Init Code:', to make sure that the entries are valid for the chosen microcontroller type.

Notice that the screen calibration can either be set as 'Manual' or 'Preset'. In this project, the 'Manual' option is chosen so that the screen can be calibrated during the run time.

3. *Add Screen and Give it a Name*
   Give a name to the screen. Let us rename the screen to MainScreen. In the Screens Properties on the left, find the Name property and change it to MainScreen, as shown in

www.it-ebooks.info

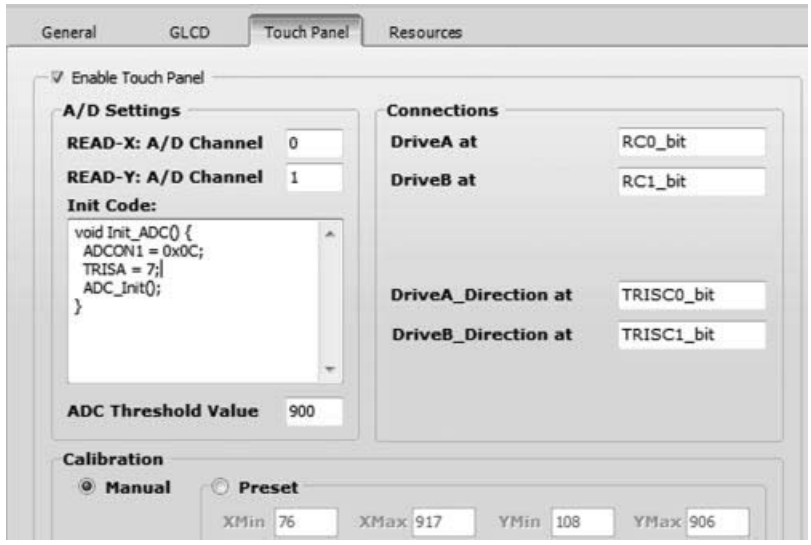


Figure 14.4 Touch Panel settings

Figure 14.5. A new screen can be added if desired, by clicking on the Add Screen icon (green '+' symbol).

4. *Place Components*
   We can now use the Tools displayed on the right-hand side to place components on our screen. Click 'Label', place it on the screen, and change its 'Caption' to 'TOGGLE LED:' in the 'Properties' window in the bottom left-hand side. Then click 'Rounded Button' and place it on the screen. Change the 'Caption' of this button to 'LED'. Figure 14.6 shows the screen layout with the components.
5. *Assign Actions to Components*
   We can now add actions to our LED component, so that when the user clicks on this component, we can toggle the LED. Select the LED component by clicking on it. Then, double click on 'OnClick' property in the Properties window. You should see the event function name 'ButtonRound1Click' displayed in the Properties window and an empty function with this name will appear in the middle of the screen. Add the code associated

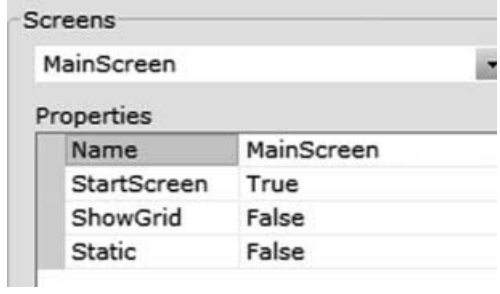


Figure 14.5 Name the screen as MainScreen

www.it-ebooks.info

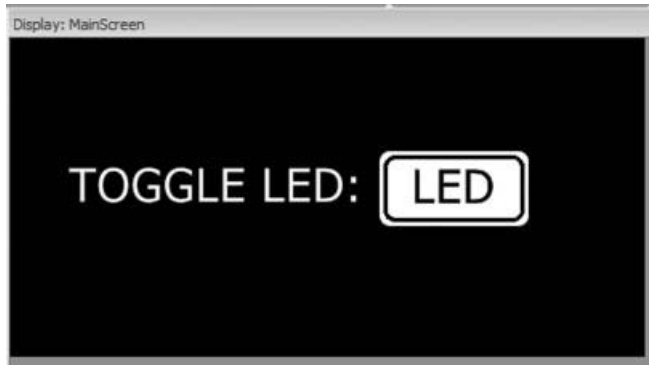


Figure 14.6 Adding Components onto the Screen

with this event. Here, we wish to toggle the LED when the LED button is clicked, therefore, enter the code shown in Figure 14.7 to the function body.

6. *Generate the Code*
   We are now ready to generate the code for our project. Just click the 'Generate Code' icon in the top menu. You should get a message to say that the code has been generated successfully. The generated code can be seen by clicking the 'Generated Code' at the bottom part of the screen.
7. *Compile the Code and Load to the Microcontroller*
   Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before by clicking the 'Build' icon in the top menu, and

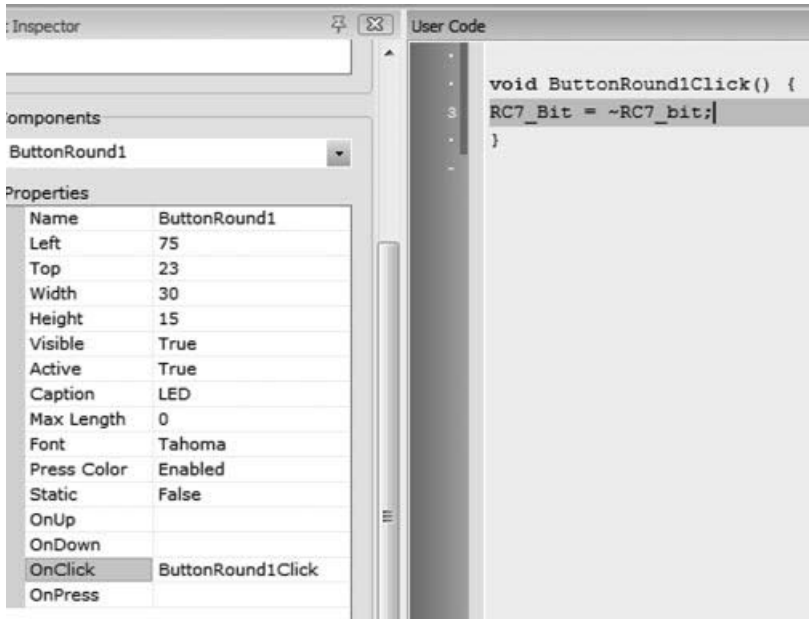


Figure 14.7 Add the code associated with the event

www.it-ebooks.info

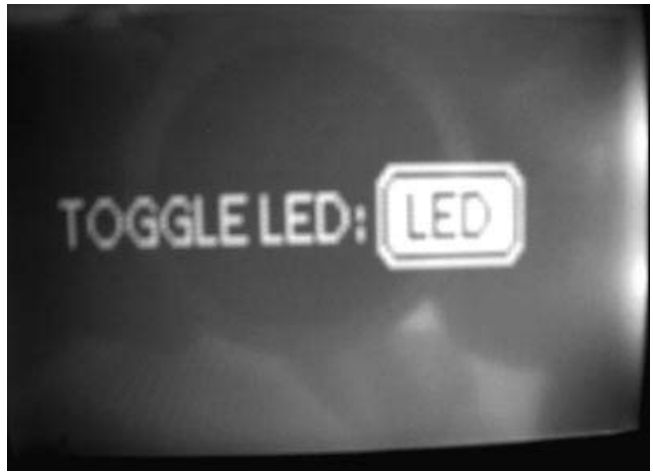


Figure 14.8 Typical display when the program is run

load the code to the target microcontroller by clicking the 'Tools -> mE Programmer' in the top menu.

You should now see the display, as in Figure 14.8. The LED connected to pin RC7 should toggle as you click the 'TOGGLE LED' button. When the program is run, the user is asked initially to calibrate the screen by touching the appropriate points of the screen in response to prompts 'TOUCH BOTTOM LEFT' and 'TOUCH UPPER RIGHT'.

The code generated by the Visual GLCD program is large and consists of three modules. Assuming the project name is TOGGLE, the following modules are generated:

- *The main program*: for example TOGGLE_main.C;
- *Events code*: for example TOGGLE_events_code.C;
- *Driver program*: for example TOGGLE_driver.C.

In addition, a number of include files are generated, for example TOGGLE_objects.h and TOGGLE_resources.h. The details of the generated files are beyond the scope of this book. Interested readers should consult the Visual GLCD documentation.

14.2 PROJECT 14.2 – Toggle more than One LED

14.2.1 Project Description

This project is similar to the previous project, but here the user is given the option of toggling more than one LED. Four upper bits of PORT C (RC4 to RC7) are used in the project, and the screen offers the following options:

- Turn OFF all LEDs RC4 to RC7.
- Turn ON all LEDs RC4 to RC7.
- Toggle individual LEDs from RC4 to RC7.

www.it-ebooks.info

Figure 14.9 Adding Components onto the Screen

14.2.2 Block Diagram

The block diagram of the project is the same as in Figure 13.3, but four LEDs are connected to upper four pins of PORT C instead of just one LED.

The operation of the project is as follows: After power-up and calibration, the user can touch the ALL ON box to turn ON all the four LEDs. Similarly, touching the All OFF box will turn OFF all the four LEDs. Touching the boxes for individual LEDs will toggle the corresponding LEDs.

14.2.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 13.5. A standard 128×64 pixel GLCD is used in this project. Switching transistors are used as the touch screen controller. A PIC18F45K22 type microcontroller is used with an 8 MHz crystal.

The project creation and configuration are as in the previous project. The project is named TOGGLEALL. The components are placed on the screen, as shown in Figure 14.9. ALL ON, ALL OFF boxes are shown as 'Rounded Buttons'. The LEDs are shown as 'Circles'.

The contents of the user created events file are shown in Figure 14.10. Notice how the events are created as functions.

When the program is run, initially the screen is calibrated, and then the various shapes are shown on the display, as in Figure 14.11. Clicking box ALL ON will turn ON all four upper LEDs of PORT C. Similarly, clicking ALL OFF will turn OFF all four upper LEDs of PORT C. Individual bits of the upper four LEDs can be toggled by clicking on the appropriate bit number.

14.3 PROJECT 14.3 – Mini Electronic Organ

14.3.1 Project Description

In this project, a mini electronic organ is designed. The organ has 8 keys corresponding to the musical octave starting with $A_4 = 440$ Hz. A buzzer is used to generate the musical notes. Pressing a note key on the screen plays the note with the correct frequency.

www.it-ebooks.info

```
#include "TOGGLEALL_objects.h"
#include "TOGGLEALL_resources.h"

//--------------------- User code ---------------------//

//----------------- End of User code ------------------//

// Event Handlers

void ButtonRound1Click() {
PORTC = PORTC | 0xF0;          // Turn ON upper 4 LEDs
}

void ButtonRound2Click() {
PORTC = PORTC & 0x0F;      // Turn OFF upper 4 LEDs
}

void CircleButton4Click() {
RC4_bit = ~RC4_bit;
}

void CircleButton1Click() {
RC5_bit = ~RC5_bit;
}

void CircleButton3Click() {
RC6_bit = ~RC6_bit;
}

void CircleButton2Click() {
RC7_bit = ~RC7_bit;
}
```

Figure 14.10 Contents of the events file

Figure 14.11 Typical display when the program is run

www.it-ebooks.info

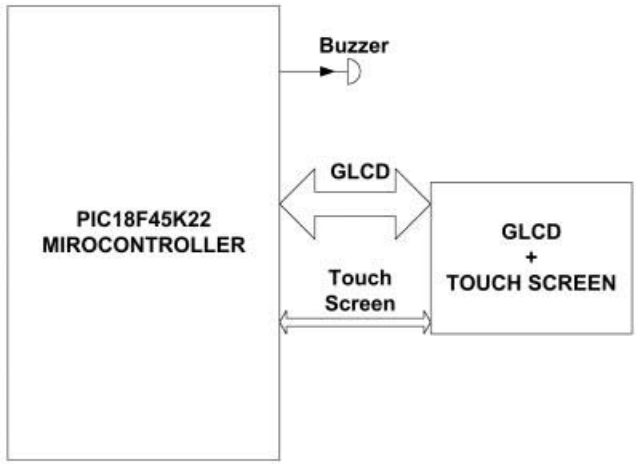


Figure 14.12 Block diagram of the project

14.3.2 Block Diagram

The block diagram of the project is shown in Figure 14.12.

The operation of the project is as follows: After power-up and the calibration, the user can touch the required note keys to play simple music.

14.3.3 Circuit Diagram

The circuit diagram of the project is as shown in Figure 14.13. A standard 128×64 pixel GLCD is used in the project. A buzzer is connected to port pin RC2 of the microcontroller through an NPN switching transistor. Switching transistors are used as the touch screen controller. A PIC18F45K22 type microcontroller is used with an 8 MHz crystal. If you are using the EasyPIC 7 development board, then jumper J12 of the buzzer should be connected to RC2.

The Visual GLCD configuration settings are slightly different, as shown in Figure 14.14. Here, the mikroC Pro for PIC sound library is initialised in the General settings.

The project creation and configuration are as in the previous project. The project is named GLCDMUSIC. The components are placed on the screen, as shown in Figure 14.15, in the form of a musical keyboard, using the 'Rounded Button' tools. The caption of each key is changed via the Properties window.

The frequencies of the musical notes in the A_4 octave are given in Table 14.1. The contents of the user created events file are shown in Figure 14.16. Notice how the events are created here as functions. The 'OnPress' action is used to detect the key presses. You should generate code using the Visual GLCD 'Generate Code' button, and then start the compiler by clicking 'Start Compiler' button to compile the program.

The mikroC Pro for PIC sound library function Sound_Play is used to generate sound with the buzzer. This function has two arguments: the first argument is the frequency of the tone to be generated, and the second argument is the duration in milliseconds. Notice that the frequency can only take integer values. Before compiling the project, make sure that the sound

www.it-ebooks.info

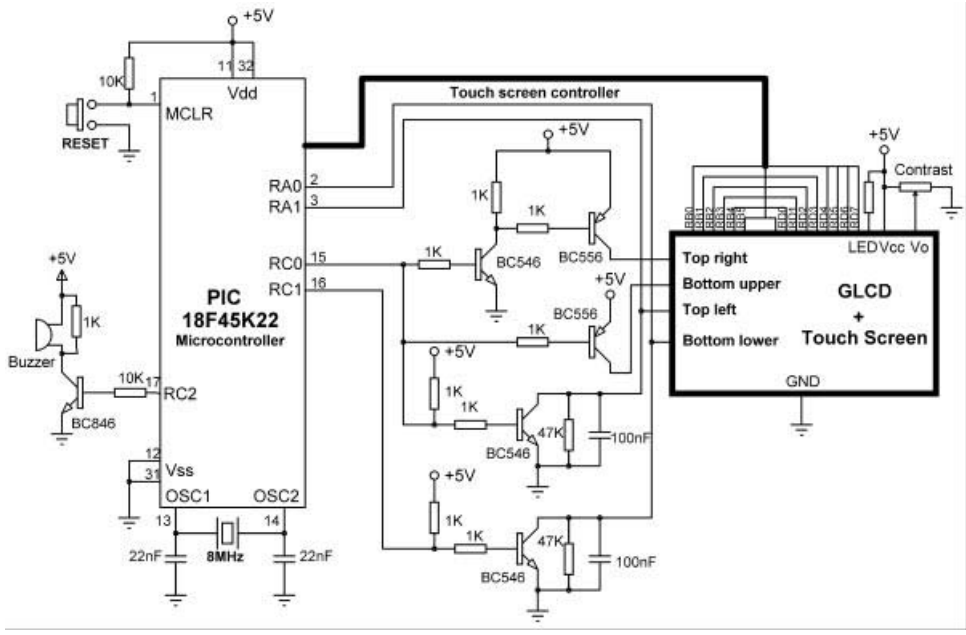


Figure 14.13 Circuit diagram of the project

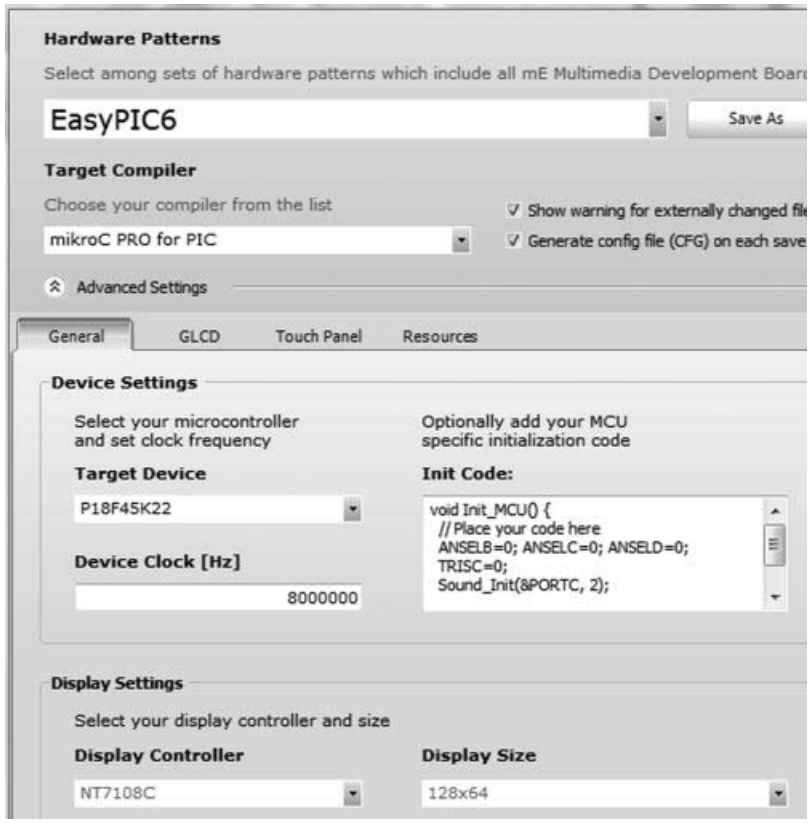


Figure 14.14 General settings

www.it-ebooks.info

Figure 14.15 Adding Components onto the Screen

Table 14.1 Frequencies of musical notes

Musical Note	Frequency (Hz)
C	261.63
D	293.66
E	329.63
F	349.23
G	392.00
A	440.00
B	493.88
C	523.25

library is included in the compilation process. To check this, click the View -> Library Manager at the top menu, and make sure that the 'Sound' library is ticked.

When the program is run, initially the screen is calibrated, and then the musical keyboard shapes are displayed on the screen, as shown in Figure 14.17. Only one octave is considered in this project for simplicity. Simple music can be played using the keyboard.

Notice that you may need to include the 'Sound' library in your project if you get compilation errors. Select View -> Library Manager in mikroC Pro for PIC and tick the sound library to include this library.

14.4 PROJECT 14.4 – Using the SmartGLCD

14.4.1 Project Description

SmartGLCD is a 240×128 pixel microcontroller based resistive touch screen graphics development tool (see Chapter 4), with built-in microcontroller, designed and developed by mikroElektronika (http://www.mikroe.com). The main advantage of using SmartGLCD is that everything for development is included on a 14×9 cm PCB. SmartGLCD is based on the PIC18F8722 microcontroller, which is loaded with Bootloader software, so that the

www.it-ebooks.info

```
#include "GLCDMUSIC_objects.h"
#include "GLCDMUSIC_resources.h"

//---------------------- User code ----------------------//

//----------------- End of User code ------------------//

// Event Handlers

void ButtonRound1Press() {                    // Note C (261.63 Hz)
Sound_Play(261, 250);
}


void ButtonRound2Press() {                    // Note D (293.66 Hz)
Sound_Play(293,250);
}

void ButtonRound3Press() {                    // Note E (329.63 Hz)
Sound_Play(329, 250);
}

void ButtonRound4Press() {                    // Note F (349.23 Hz)
Sound_Play(349, 250);
}

void ButtonRound5Press() {                    // Note G (392 Hz)
Sound_Play(392, 250);
}

void ButtonRound6Press() {                    // Note A (440 Hz)
Sound_Play(440, 250);
}

void ButtonRound7Press() {                    // Note B (493.88 Hz)
Sound_Play(494, 250);
}

void ButtonRound8Press() {                    // Note C (523.25 Hz)
Sound_Play(523, 250);
}
```

Figure 14.16 Contents of the events file

device can be programmed easily without the need for expensive programmers. The device contains USB UART module for RS232 based serial communication. In addition, a microSD card interface is provided. SmartGLCD is powered via its USB port. The device provides a large number of I/O pads for easy expansion.

In this project, a mini RS232 soft keyboard for serial communication is developed using the Visual GLCD software. All the numeric and alphabetic keys are provided in the design. The project consists of two screens. The first screen is used to configure the communications speed (i.e. Baud rate). The second screen is the soft keyboard. Pressing a key on the keyboard will send the ASCII code of that key via the serial USB UART port of the SmartGLCD.

www.it-ebooks.info

Figure 14.17 Typical display when the program is run

14.4.2 Block Diagram

The block diagram of the project is as shown in Figure 14.18.

The operation of the project is as follows: After power-up and screen calibration, the first screen is displayed where the user is asked to select the required Baud rate by touching the appropriate box on the screen. Then, the keyboard is displayed in the second screen.

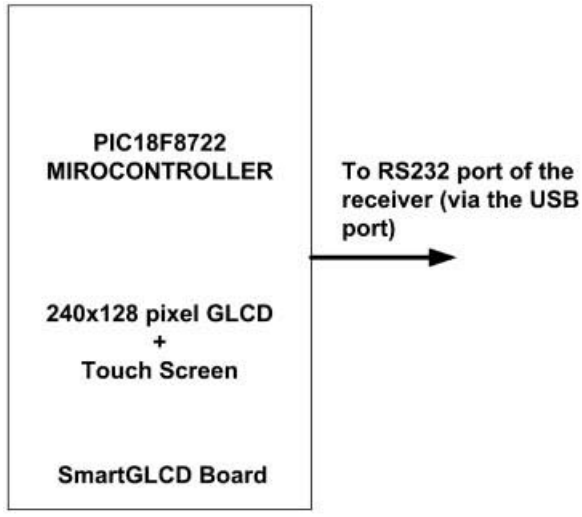


Figure 14.18 Block diagram of the project

www.it-ebooks.info

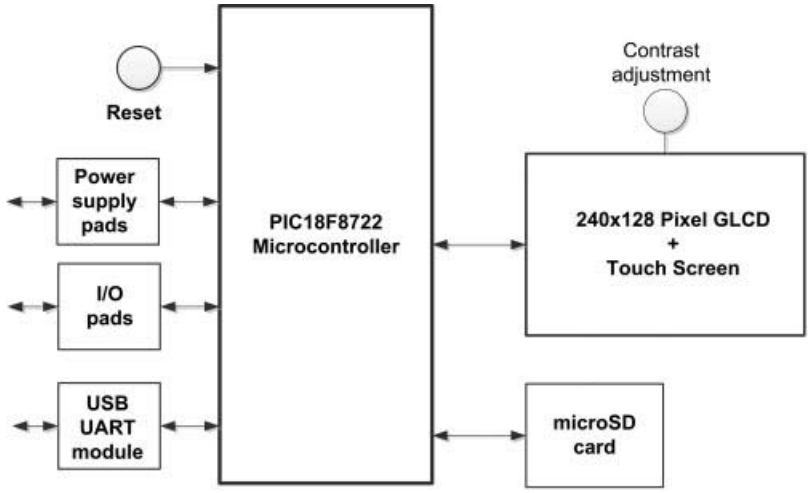


Figure 14.19 Circuit diagram of the SmartGLCD as blocks

14.4.3 Circuit Diagram

The circuit diagram of the SmartGLCD board is shown in Figure 14.19, in the form of blocks. A PIC18F8722 type microcontroller is used in the design. Details of the actual circuit diagram can be obtained from the manufacturer's Web site (http://www.mikroe.com).

The steps in creating the project using the Visual GLCD software are given below:

1. *Create Project File*
   Start the Visual GLCD software as before. Select Project -> New from the top-down menu and enter the project name. In this project, the name TERMINAL is given to the project.
2. *Configure Project*
   The General settings are shown in Figure 14.20. Notice that when the SmartGLCD is selected from the hardware patterns, all the parameters associated with this board are configured automatically.
   The GLCD settings are shown in Figure 14.21.
   The Touch Panel settings are shown in Figure 14.22. The calibration is selected as 'Manual', so that the screen can be calibrated during run time.
3. *Add Screen and Give it a Name*
   The project consists of two screens: 'Screen1' is the 'StartScreen', so that this screen is first displayed when the program is run. The second screen is named 'Screen2', and this screen is displayed when the user clicks OK after selecting the Baud rate.
4. *Place components*

14.4.3.1 Screen1

'Screen1' is used to select the Baud rate between 2400, 4800, 9600 and 19 200. The selection shapes are designed using the 'Circle Button' tools and the caption of each shape is set using the Properties window. Figure 14.23 shows the 'Screen1' layout.

www.it-ebooks.info

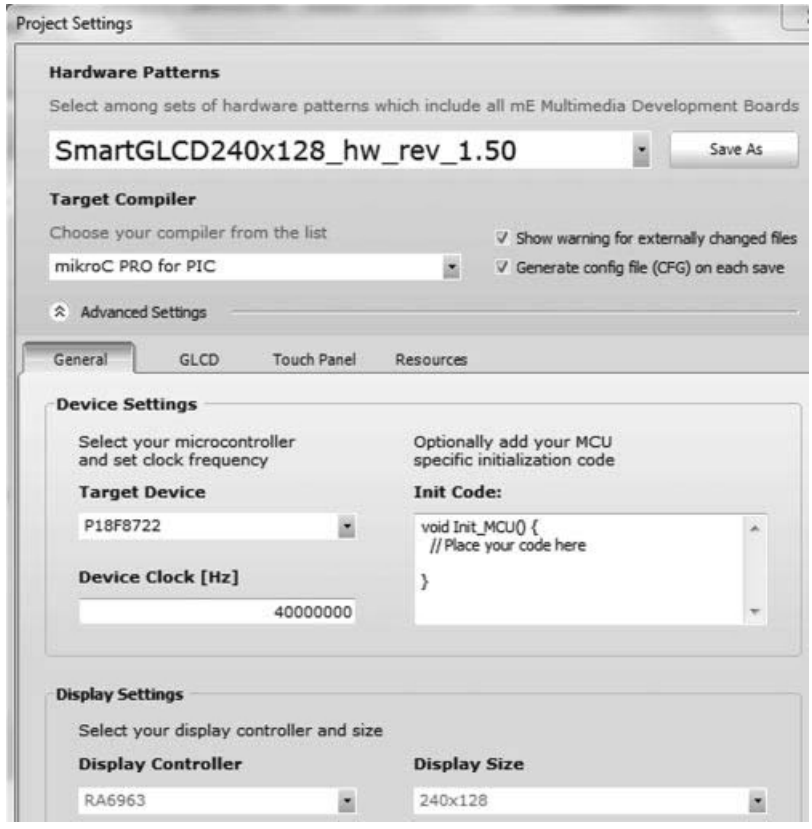


Figure 14.20 General settings

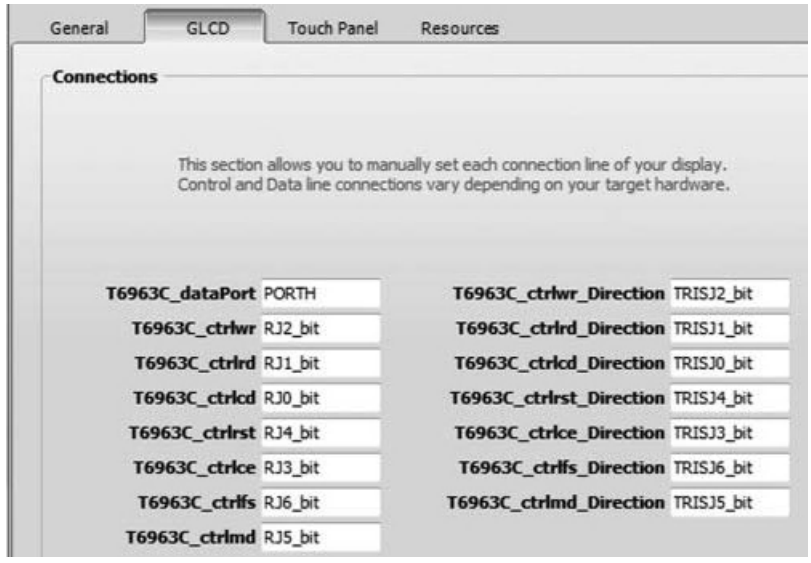


Figure 14.21 GLCD settings

www.it-ebooks.info

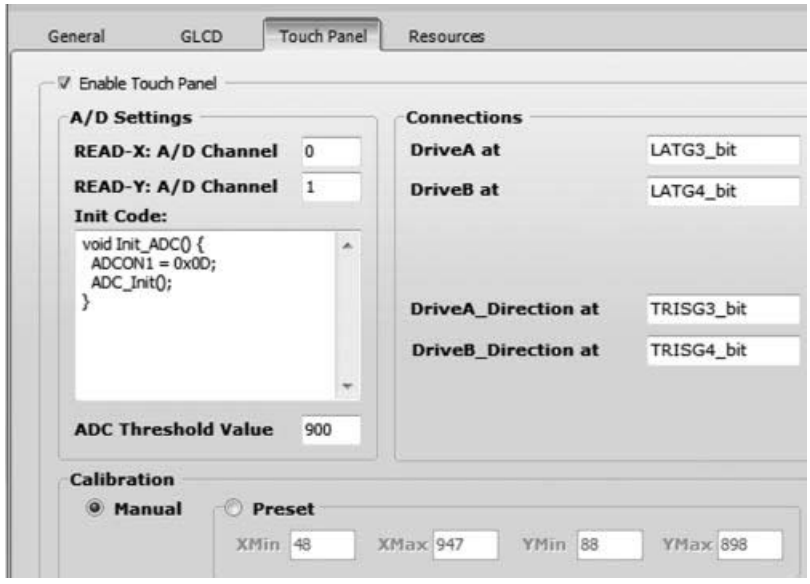


Figure 14.22 Touch Panel settings

14.4.3.2 Screen2

'Screen2' is the keyboard and is made up using the 'Button' tools. The screen layout is designed with the help of the 'Align and Distribute' tool. To use this tool, let us say that we want to make the first row of the keyboard with 11 keys. We draw one key and then duplicate it 10 times. Then place the first one and the last one in the right positions.Then, we select all the 11 keys, right click the mouse and select the 'Align and Distribute' option. Select the following options to align all the keys correctly:

- Align Top;
- Make Same Width;

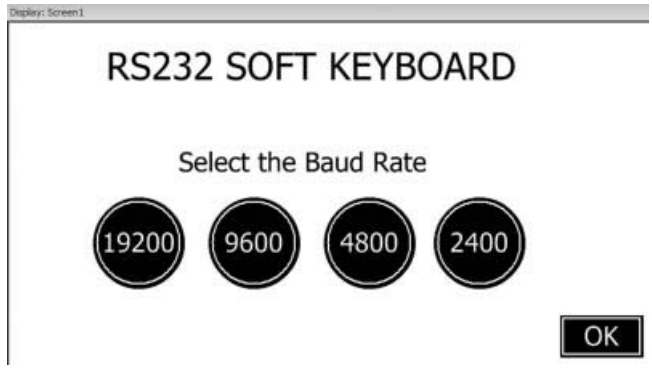


Figure 14.23 Screen1 is used to select the Baud rate

www.it-ebooks.info

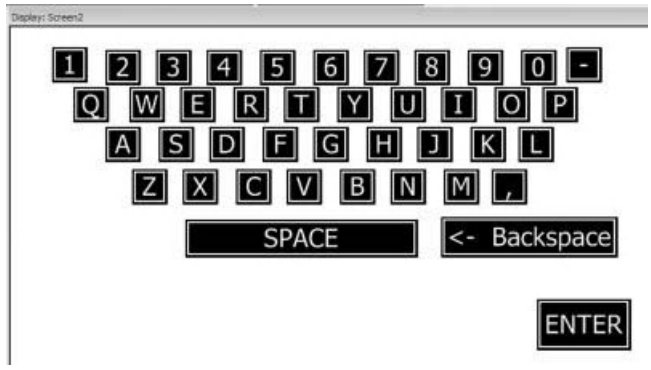


Figure 14.24 Screen2 layout

- Make Same Height;
- Space Equally Horizontal;
- Space Equally Vertical.

Then, do the same for the other rows. After all the keys are placed correctly, you should change their captions. Figure 14.24 shows the Screen2 layout.

5. *Assign Actions to Components*
   We can now add actions to our components, so that when the user clicks on a component, the function that handles the component is activated.

14.4.3.3 Screen1

Select the shape 2400 and double click the 'OnClick' option in the Properties window. An empty function will be displayed. Enter the following code inside the body of the function to set the Baud rate to 2400:

```
UART1_Init(2400);                    // 2400 Baud
```

Repeat for all the other Baud rates. Enter the following code for the OK button, so that when the button is clicked, Screen2 will be displayed:

```
Drawscreen(&Screen2);                // Draw Screen2
```

Figure 14.25 shows the user code for Screen1.

14.4.3.4 Screen2

Select shape with caption 1, and double click the 'OnClick' option in the properties window. Enter the following code inside the body of the function, so that character '1' is sent to the UART port when the key is pressed:

```
UART1_WRITE('1');                    // Key 1
```

www.it-ebooks.info

```
#include "Terminal_objects.h"
#include "Terminal_resources.h"

//--------------------- User code ---------------------//

//----------------- End of User code ------------------//

// Event Handlers

void Button1Click() {
Drawscreen(&Screen2);                          // Draw the second screen
}

void CircleButton1Click() {
  UART1_Init(19200);                            // 19200 Baud
}

void CircleButton2Click() {
  UART1_Init(9600);                             // 9600 Baud
}

void CircleButton3Click() {
  UART1_Init(4800);                             // 4800 Baud
}

void CircleButton4Click() {
  UART1_Init(2400);                             // 2400 Baud
}
```

Figure 14.25 User code for Screen1

Repeat for all the other keys. Figure 14.26 shows the user code for Screen2 (notice that the user codes for Screen1 and Screen2 are combined in a single file).

6. *Generate the Code*
   We are now ready to generate the code for our project. Just click the 'Generate Code' icon in the top menu. You should get a message to say that the code has been generated successfully. The generated code can be seen by clicking the 'Generated Code' at the bottom part of the screen.
7. *Compile the Code and Load to the Microcontroller*

Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before, by clicking the 'Build' icon in the top menu.

The microcontroller on the SmartGLCD board is supplied with a Bootloader software that enables the device to be programmed directly from a PC. Before programming, the SmartGLCDBootloader software is required on the PC. The steps to program the SmartGLCD board are given below:

- Download and install the SmartGLCD software from the manufacturer's Web site http://www.mikroe.com/eng/products/view/443/smartglcd-240x128-board/.
- Connect the SmartGLCD board to the PC via the USB port.

www.it-ebooks.info

```
void Button2Click() {                // Key 1
  UART1_WRITE('1');
}

void Button3Click() {                // Key 2
  UART1_WRITE('2');
}

void Button4Click() {                // Key 3
  UART1_WRITE('3');
}

void Button5Click() {                // Key 4
  UART1_WRITE('4');
}

void Button6Click() {
  UART1_WRITE('5');                  // Key 5
}

void Button7Click() {                // Key 6
  UART1_WRITE('6');
}

void Button8Click() {                // Key 7
  UART1_WRITE('7');
}

void Button9Click() {                // Key 8
  UART1_WRITE('8');
}

void Button10Click() {               // Key 9
  UART1_WRITE('9');
}

void Button11Click() {               // Key 0
  UART1_WRITE('0');
}

void Button12Click() {               // Key -
  UART1_WRITE('-');
}

voidButtonQClick() {                 // Key Q
   UART1_WRITE('Q');
}

voidButtonWClick() {                  // Key W
  UART1_WRITE('W');
}
```

Figure 14.26 User code for Screen2

www.it-ebooks.info

```
voidButtonEClick() {                    // Key E
  UART1_WRITE('E');
}

voidButtonRClick() {                    // Key R
  UART1_WRITE('R');
}

voidButtonTClick() {                    // Key T
  UART1_WRITE('T');
}

voidButtonYClick() {                    // Key Y
  UART1_WRITE('Y');
}

voidButtonUClick() {                    // Key U
  UART1_WRITE('U');
}

voidButtonIClick() {                    // Key I
  UART1_WRITE('I');
}

voidButtonOClick() {                    // Key O
  UART1_WRITE('O');
}

voidButtonPClick() {                    // Key P
  UART1_WRITE('P');
}

voidButtonAClick() {                    // Key A
  UART1_WRITE('A');
}

voidButtonSClick() {                    // Key S
  UART1_WRITE('S');
}

voidButtonDClick() {                    // Key D
  UART1_WRITE('D');
}

voidButtonFClick() {                    // Key F
  UART1_WRITE('F');
}

voidButtonGClick() {                    // Key G
  UART1_WRITE('G');
```

Figure 14.26 (*Continued*)

www.it-ebooks.info

```
}

voidButtonHClick() {                    // Key H
  UART1_WRITE('H');
}

voidButtonJClick() {                    // Key J
  UART1_WRITE('J');
}

voidButtonKClick() {                    // Key K
  UART1_WRITE('K');
}

voidButtonLClick() {                    // Key L
  UART1_WRITE('L');
}

voidButtonZClick() {                    // Key Z
  UART1_WRITE('Z');
}

voidButtonXClick() {                    // Key X
  UART1_WRITE('X');
}

voidButtonCClick() {                    // Key C
  UART1_WRITE('C');
}

voidButtonVClick() {                    // Key V
  UART1_WRITE('V');
}

voidButtonBClick() {                    // Key B
  UART1_WRITE('B');
}

voidButtonNClick() {                    // Key N
  UART1_WRITE('N');
}

voidButtonMClick() {                    // Key M
  UART1_WRITE('M');
}

voidButtonCOMMAClick() {                // Key,
  UART1_WRITE(',');
}

voidButtonSPACEClick() {                // Key Space
```

Figure 14.26 (*Continued*)

www.it-ebooks.info

```
  UART1_WRITE(' ');
}

voidButtonENTERClick() {              // Key Enter
  UART1_WRITE(0X0D);
  UART1_WRITE(0X0A);
}

voidButtonBCKSPCClick() {             // Key Backspace
  UART1_WRITE(0X08);
}
```

Figure 14.26 (*Continued*)

- Start the SmartGLCDBootloader software on the PC (see Figure 14.27).
- Identify the COM port used by the SmartGLCDBootloader. Go to Control Panel ->Device Manager, click icon 'Ports(COM & LPT)' and check the 'USB Serial Port' number. In Figure 14.28, the port number is 15.
- Click 'Change Settings' button of the SmartGLCDBootloader and set the 'Port' to the number found in the above step.
- Press the RESET button on the SmartGLCD board. Then click the 'Connect' button within 5 seconds. You should see the message 'Connected . . . '. The microcontroller on the SmartGLCD board can now be programed.
- Click 'Browse for HEX' and select the '.hex' file of your project.

Figure 14.27 The SmartGLCD Bootloader software

www.it-ebooks.info

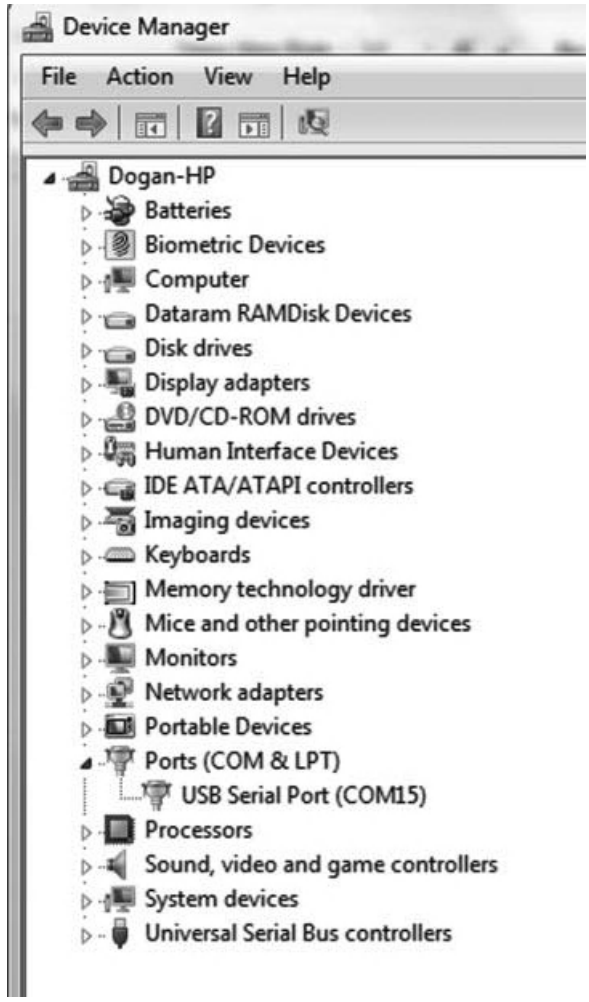


Figure 14.28 Identifying the COM port number

- Click 'Begin Uploading' to program the microcontroller. You should now see a progress bar as the programming is in progress. When the uploading is finished, click 'OK' to exit.
- Reset the SmartGLCD board. The display will start after 5 seconds (if during the first 5 seconds after a RESET there is no communication with a PC, then the microcontroller assumes that this is a normal run and not a programming run).

When the program is run, the user is asked initially to calibrate the screen by touching the bottom-left and upper-right points of the screen. Figure 14.29 shows Screen1 displayed on the SmartGLCD. Select the Baud rate as 9600 by touching its button. Then click 'OK' to display Screen2, as shown in Figure 14.30.

Now, when a key is touched on the keyboard, its ASCII code is sent to the receiving serial device connected to the SmartGLCD board.

www.it-ebooks.info

Figure 14.29 Displaying Screen1

14.4.3.5 Testing the Program

The program can easily be tested by connecting the SmartGLCD board to a device, which can accept RS232 serial data from its USB port. Perhaps the easiest test is by using a PC. The mikroC Pro for PIC compiler includes a USART Terminal that can be selected from the Tools. The steps for testing the program are given below:

- Connect the SmartGLCD board to the USB port of the PC.
- Calibrate the device by touching the appropriate points of the screen.
- Select the Baud rate as 9600 and click OK to see the keyboard on the next screen.
- Start the mikroC Pro for PIC compiler.
- Click Tools -> USART Terminal. You should see the serial communications program activated.

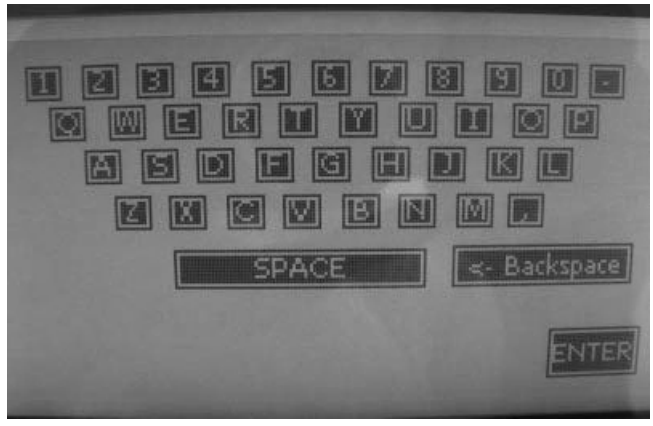


Figure 14.30 Displaying Screen2

www.it-ebooks.info

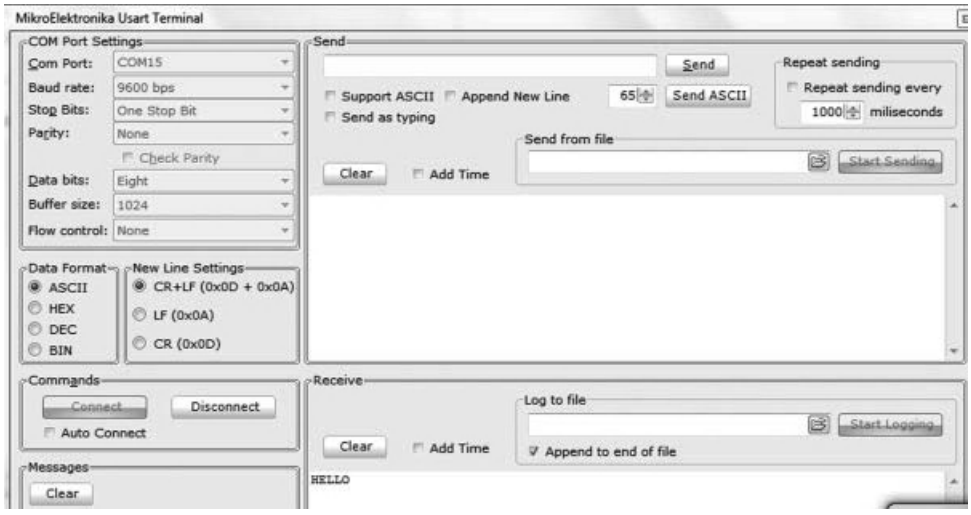


Figure 14.31 ThemikroC Pro for PIC USART Terminal

- Make sure that the Baud rate is selected as 9600 in 'COM Port Settings'. Click 'Connect' in the 'Commands' window to connect to the serial port.
- Touch any key on the SmartGLCD keyboard. You should see the touched keys displayed in the 'Receive' window of the USART Terminal. Figure 14.31 shows the text 'HELLO' displayed.

14.5 PROJECT 14.5 – Decimal to Hexadecimal Converter using the SmartGLCD

14.5.1 Project Description

In this project we shall be converting decimal numbers into hexadecimal format by designing a calculator program using the Visual GLCD software and the SmartGLCD module. A keypad will be designed with decimal numbers 0 to 9, so that a decimal number can be entered. Touching the ENTER button will convert the entered number into hexadecimal and display it. The CLR button is used to clear the screen, so that a new number can be entered.

14.5.2 Screen Layout

The required screen layout is shown in Figure 14.32.

14.5.3 Circuit Diagram

The circuit diagram of the SmartGLCD board is as shown in Figure 14.19 in the form of blocks. A PIC18F8722 type microcontroller is used in the design. Details of the actual circuit diagram can be obtained from the manufacturer's Web site (http://www.mikroe.com).

www.it-ebooks.info

Figure 14.32 Screen layout of the project

The steps in creating the project using the Visual GLCD software are given below:

1. *Create Project File*
   Start the Visual GLCD software as before, and name the project as CONVERTER.
2. *Configure Project*
   The settings are as in the previous project (see Figures 14.20, 14.21, and 14.22).
3. *Add Screen and Give it a Name*
   There is only one screen in the project. Name the screen as 'Converter'.
4. *Place components*
   We can now place the components on the screen. Let us use the Button components for the keys. Click the Button in Tools window and place the keys on the screen, including the CLR and ENTER keys. Change the key fonts, as shown in Figure 14.33. Change names of CLR and ENTER keys to Buttonclr and Buttonenter, respectively. Place a Rounded Button for the display and change its name to DispRes and its caption to Dec-Hex. Finally, use the Line component to draw lines around the calculator.
5. *Assign Actions to Components*
   We can now add actions to our components, so that when the user clicks on a component, the function that handles the component is activated. Select key labelled 1 by clicking on it, then double click on 'OnClick' in the properties window to see the event code corresponding to this key. Enter the following code for this key. This code calculates the total value of the number entered so far and stores in variable *sum*. Character array *Txt* is loaded with character '1' and this is converted into a string by NULL terminating it. Then the built-in function strcat is used to copy this character to the caption of button DispRes, which shows the character in the display of the calculator. Function DrawButton(&DispRes) redraws the display button.

www.it-ebooks.info

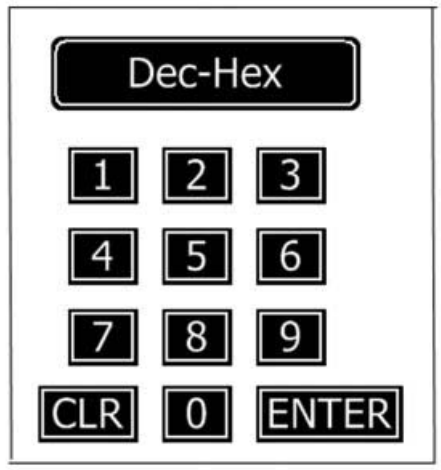


Figure 14.33 Components placed on the Visual GLCD screen

```
void Button1Click() {
char Txt[2];
sum = 10*sum + 1;
Txt[0] = '1';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}
```

Repeat the above process for all the keys 0 to 9. Then, select key CLR, double click the 'OnClick' in the Properties window and enter the following code for the clear key. This code clears the screen by copying an empty string to the display and then clearing variable *sum*.

```
voidButtonclrClick() {
strcpy(DispRes.caption, "");
sum = 0;
DrawButton(&DispRes);
}
```

Select the ENTER key and enter the following code for its event function. When the ENTER key is touched, function ButtonenterClick is invoked. This function clears a variable called *q*, clears the display, and calls to function PrintOut to convert variable *sum* (which is the total number entered by the user) from decimal into hexadecimal. Function PrintOut is a built-in function, which can be used to convert between different

www.it-ebooks.info

number formats. The function calls a printHandler function, which must be defined before it is called. String (, '%04X', sum) converts sum into a 4 digit hexadecimal number. Entering a '0' at the beginning forces the converted number to be padded with 0s if it is less than 4 digits. Thus, for example, if *sum* is decimal 25, the converted hexadecimal number will be '0019'. The digits of the converted number are stored in character array *Txt*. After receiving the four hexadecimal characters, array *Txt* is terminated with a NULL character to make it a string, and this string is shown in the display part of the calculator. Another conversion can be done after clearing the screen by touching the CLR key.

```
voidPrintHandler(unsigned char c)
{
    Txt[q] = c;
q++;
if(q == 4)
    {
Txt[q] = 0x0;
strcpy(DispRes.caption, Txt);
DrawButton(&DispRes);
    }
}

voidButtonenterClick() {
q = 0;
strcpy(DispRes.caption, "");
DrawButton(&DispRes);
PrintOut(PrintHandler, "%04X",sum);
}
```

Notice that variables *q*, *sum* and *Txt* are global and they must be defined at the beginning of the user code. The complete user code is shown in Figure 14.34.

6. *Generate the Code*
   We are now ready to generate the code for our project. Just click the 'Generate Code' icon in the top menu. You should get a message to say that the code has been generated successfully. The generated code can be seen by clicking the 'Generated Code' at the bottom part of the screen.
7. *Compile the Code and Load to the Microcontroller*

Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before, by clicking the 'Build' icon in the top menu. Before compiling the program, make sure that the string library is included in the project. To check this, click View -> Library Manager and check 'C_String', if it is not already checked.

Load the program to the microcontroller on the SmartGLCD module using the Bootloader, as described in the previous project. After the program is loaded, press the Reset key. The program should start after about 5 seconds. You should calibrate the screen by

www.it-ebooks.info

```
#include "Converter_objects.h"
#include "Converter_resources.h"

//--------------------- User code ---------------------//
unsignedint sum = 0;
unsigned char q;
unsigned char Txt[5];
//----------------- End of User code ------------------//

// Event Handlers
void Button1Click() {
char Txt[2];
sum = 10*sum + 1;
Txt[0] = '1';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button2Click() {
char Txt[2];
sum = 10*sum + 2;
Txt[0] = '2';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button3Click() {
char Txt[2];
sum = 10*sum + 3;
Txt[0] = '3';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button4Click() {
char Txt[2];
sum = 10*sum + 4;
Txt[0] = '4';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button5Click() {
char Txt[2];
sum = 10*sum + 5;
Txt[0] = '5';
Txt[1] = 0x0;
```

Figure 14.34 Complete user code

www.it-ebooks.info

```
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button6Click() {
char Txt[2];
sum = 10*sum + 6;
Txt[0] = '6';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button7Click() {
char Txt[2];
sum = 10*sum + 7;
Txt[0] = '7';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button8Click() {
char Txt[2];
sum = 10*sum + 8;
Txt[0] = '8';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button9Click() {
char Txt[2];
sum = 10*sum + 9;
Txt[0] = '9';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

void Button10Click() {
char Txt[2];
sum = 10*sum + 0;
Txt[0] = '0';
Txt[1] = 0x0;
strcat(DispRes.caption, Txt);
DrawButton(&DispRes);
}

voidButtonclrClick() {
strcpy(DispRes.caption, "");
```

Figure 14.34 (*Continued*)

www.it-ebooks.info

```
sum = 0;
DrawButton(&DispRes);
}

voidPrintHandler(unsigned char c)
{
    Txt[q]=c;
q++;
if(q == 4)
    {
      Txt[q] = 0x0;
strcpy(DispRes.caption, Txt);
DrawButton(&DispRes);
    }
}



voidButtonenterClick() {

 q = 0;

strcpy(DispRes.caption, "");

DrawButton(&DispRes);

PrintOut(PrintHandler, "%04X",sum);

}
```

Figure 14.34 (*Continued*)

Figure 14.35 A typical display of the screen after power-up

www.it-ebooks.info

Figure 14.36 The display when decimal number 254 is entered

touching the bottom left and upper right corners. The calculator should then be displayed and is ready for use.

Figure 14.35 shows a typical display after power-up and after the screen is calibrated. Figure 14.36 shows the display when decimal number 254 is entered, and finally, Figure 14.37 shows the display after the number is converted into hexadecimal.

Figure 14.37 The display after the number is converted into hexadecimal

www.it-ebooks.info

14.6 Summary

Visual GLCD is a standalone software package used for the development of GLCD based projects. With the help of this software, users can create screens and place drag-and-drop components on these screens. This chapter has described the design of several projects using the Visual GLCD software. The early projects are based on using the standard 128×64 pixel GLCD module. Visual GLCD provides an event driven approach to graphical user based project design. Users can touch the various components placed on the screen and then associate codes with these components. These activation codes are in the form of functions that can be written by users. The use of the SmartGLCD module has been described in the chapter. This is a standalone hardware incorporating 240×128 pixel GLCD and a PIC microcontroller. Projects are given to show how to use the SmartGLCD module in projects.

Exercises

14.1 Design a project having two rectangular shaped components on a GLCD screen. Label these components as START and STOP. Write a Visual GLCD based program, such that when the START button is pressed, a buzzer connected to port pin RC7 gives a sound at the frequency of 800 Hz, and when the STOP button is pressed the buzzer stops sounding.

14.2 Design an integer calculator using the Visual GLCD software and the SmartGLCD module. The calculator should have the numbers 0 to 9, a clear key and an ENTER key. In addition, a display should be provided to show results of a calculation. Assume the four basic mathematical operations '$+ - *$/'.

14.3 Design a free-hand screen drawing project using the Visual GLCD software and the SmartGLCD module. The user should be able to draw lines on the screen by touching and moving a stylus on the screen.

14.4 Modify the program in (3) above, so that the images created can be saved.

www.it-ebooks.info

15

Using the Visual TFT Software in Graphics Projects

Visual TFT is a standalone software package used for the development of TFT based colour graphical user interfaces in embedded devices. With the help of the software, users can create screens and place drag-and-drop components on these screens, which can be used for building complex graphical applications. The Visual TFT software generates code compatible with the mikroC Pro for PIC and similar compilers developed by mikroElektronika.

Current version (2.01) of Visual TFT supports the following features:

- 15 graphical user interface components with full colour;
- full colour multiple screens;
- automatic code generation for compilers;
- multiple font support, such as regular, bold, italic, underline and strikeout;
- support for external memory;
- 'On Press/release' procedures when objects are clicked (event driven design);
- changing properties of multiple objects at the same time;
- zoom in/out option for each screen generated;
- show/hide grid options;
- print and print preview of current screen.

The Visual TFT software must be installed before it can be used. Download the software from the mikroElektronika Web site: http://www.visualtft.com. Start up the software by double clicking on the appropriate icon. The Visual TFT IDE will appear on the screen and you are now ready to use the software (you may need to have a licence or a dongle to use the software to full capacity).

Both the Visual GLCD and Visual TFT software packages have similar user interfaces. The steps to create a graphics application using the Visual TFT are similar to those used while creating Visual GLCD applications:

1. Create project files.
2. Configure project.

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

3. Add screen and give it a name.
4. Place components on the screen and configure component properties.
5. Assign actions (or events) to components.
6. Generate the code.
7. Compile the code and load to the target microcontroller.

Several examples are given in this chapter, to show how the Visual TFT software can be used in graphical projects. The first project gives all the steps in detail, while the other projects simply describe the important points in each project.

The projects described in this chapter are based on the graphics development board called 'MikroMMB board for PIC18FJ', developed by mikroElektronika (http://www.mikroe.com). This is a 8×6 cm small graphics development board (see Chapter 4), with a built-in 320×240 pixel colour TFT display on one side, and a PCB with components on the other side. The board includes everything necessary for creating powerful TFT based graphics applications. The key features of the board are:

- TFT 320×240 pixel display with resistive touch panel;
- Display capable of showing data in 262 000 different colours;
- PIC18F87J50 microcontroller;
- 8 MHz Crystal oscillator;
- microSD card slot;
- Stereo MP3 coder/decoder (VS1053);
- 3.5 mm headphone connector;
- USB mini connector;
- 8 Mbit serial flash memory (M25P80);
- Reset button;
- three axis Accelerometer (ADXL345);
- I/O connection pads.

The board is powered from the USB port and on-board voltage regulators ensure the appropriate voltage levels to each part of the board. A power LED indicates the presence of a power supply. The board can also be powered using a Li-Polymer battery, via an on-board battery connector. The microcontroller on the board can be programmed using the Bootloader software loaded to the microcontroller, or external programmers can be used (e.g. mikroProg or ICD2/3).

In this chapter, we will be using the Bootloader software to load our programs to the microcontroller on the MikroMMB board. Load and install the PC end of the Bootloader software from the Web site:

http://www.mikroe.com/eng/products/view/585/mikromedia-for-pic18fj/

15.1 PROJECT 15.1 – Countdown Timer

15.1.1 Project Description

This project describes the design of a countdown timer using the MikroMMB board for PIC18FJ (from now onwards, this board will be called the MikroMMB board). The screen

www.it-ebooks.info

consists of a soft keypad with numbers 0 to 9, a START button, a CLR button and a display box. The operation of the project is as follows: the user enters a starting number and clicks the START button. The display counts down in 1 second intervals until the count reaches zero and then it stops. Clicking the CLR button clears the display and the system is ready for the next count.

15.1.2 Block Diagram

The block diagram of the MikroMMB board is shown in Figure 15.1. A detailed circuit diagram can be obtained from the manufacturer's product guide.

The steps for creating this project are given below:

1. *Create Project File*
   Start the Visual TFT software. Select Project -> New from the top-down menu. Enter the project name and choose the desired project path. Then click OK. The name STOPWATCH is given to the project.
2. *Configure Project*
   In the Project Settings window, make the following settings:

15.1.2.1 General (see Figure 15.2)

- *Hardware patterns*: MikroMMB_for_PIC18FJ_hw_rev_1.05;
- *Target Compiler*: mikroC Pro for PIC PRO for PIC;

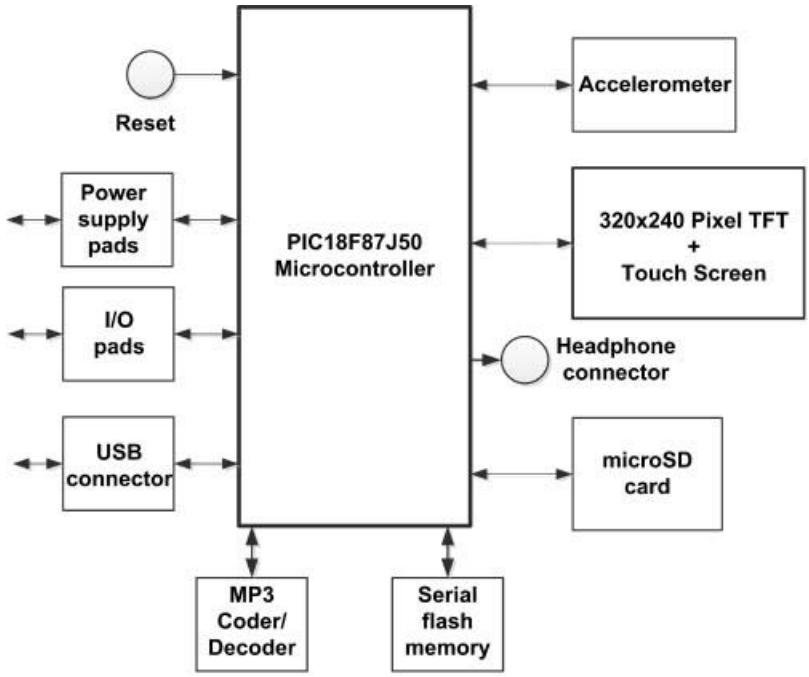


Figure 15.1 Block diagram of the MikroMMB board

www.it-ebooks.info

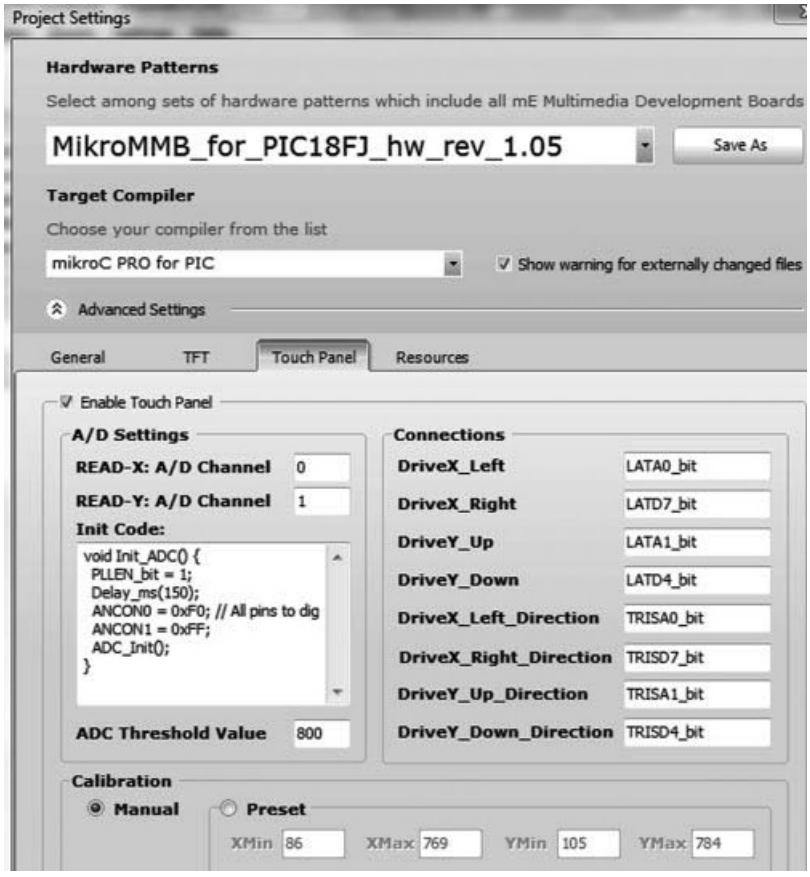


Figure 15.2 General settings

- *Target Device*: PIC18F87J50;
- *Device Clock (Hz)*: 48 000 000.

15.1.2.2 TFT (see Figure 15.3)

• TFT_Data_Port:	PORTJ	TFT_DataPort_Direction:	TRISJ
• TFT_RST:	LATD4_bit	TFT_RST_Direction:	TRISD4_bit
• TFT_BLED:	LATH5_bit	TFT_BLED_Direction:	TRISH5_bit
• TFT_RS:	LATH6_bit	TFT_RS_Direction:	TRISH6_bit
• TFT_CS:	LATG3_bit	TFT_CS_Direction:	TRISG3_bit
• TFT_RD:	LATH1_bit	TFT_RD_Directiion:	TRISH1_bit
• TFT_WR:	LATH2_bit	TFT_WR_Direction:	TRISH2_bit

www.it-ebooks.info

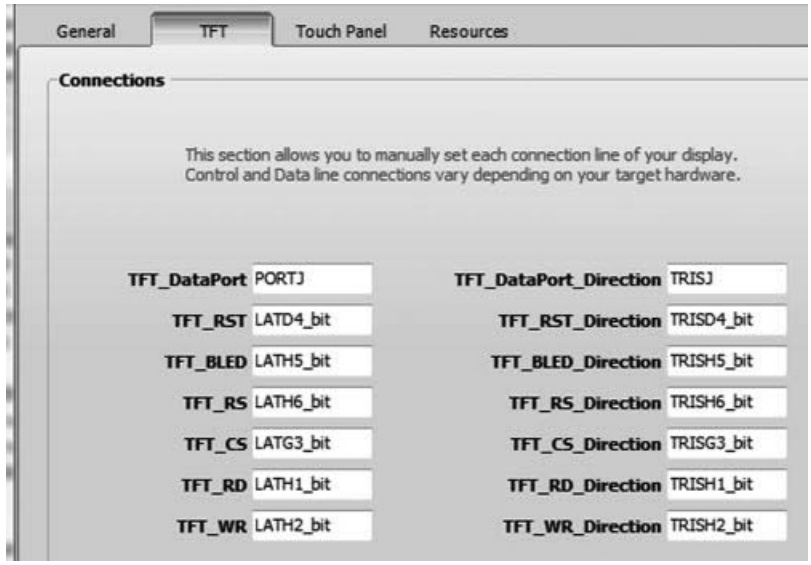


Figure 15.3 TFT settings

15.1.2.3 Touch Panel (see Figure 15.4)

```
READ-X: A/D Channel: 0
READ-Y: A/D Channel: 1

DriveX_Left:     LATA0_bit    DRIVEX_Left_Direction:   TRISA0_bit
DriveY_Right:    LATD7_bit    DriveY_Right_Direction:  TRISD7_bit
DriveY_Up:       LATA1_bit    DriveY_Up_Direction:     TRISA1_bit
DriveY_Down:     LATD4_bit    DriveY_Down_Direction:   TRISD4_bit
```

Notice that the screen calibration can either be set as 'Manual' or 'Preset'. In this project, the 'Manual' option is chosen so that the screen can be calibrated during the run time.

3. *Add Screen and Give it a Name*
   Give a name to the screen. Let us rename the screen to Screen1. In the Screens Properties on the left, find the Name property and change it to Screen1. Set the colour to white and orientation to portrait.
4. *Place components*
   We can now use the Tools displayed on the right-hand side to place components on our screen. Click 'Label', place it on the screen, and change its 'Caption' to 'COUNTDOWN TIMER' in the 'Properties' window in the bottom left-hand side. Then place the display on the screen by clicking 'Rounded Button' and position it, as shown in Figure 15.5. Change the name of this button to 'disp'. Place the keypad on the screen using the 'Rounded Button' tools. Change the names of the buttons to reflect the numbers they represent. Thus, name number 0 as 'no0', number 1 as 'no1', and so on. Place the START

www.it-ebooks.info

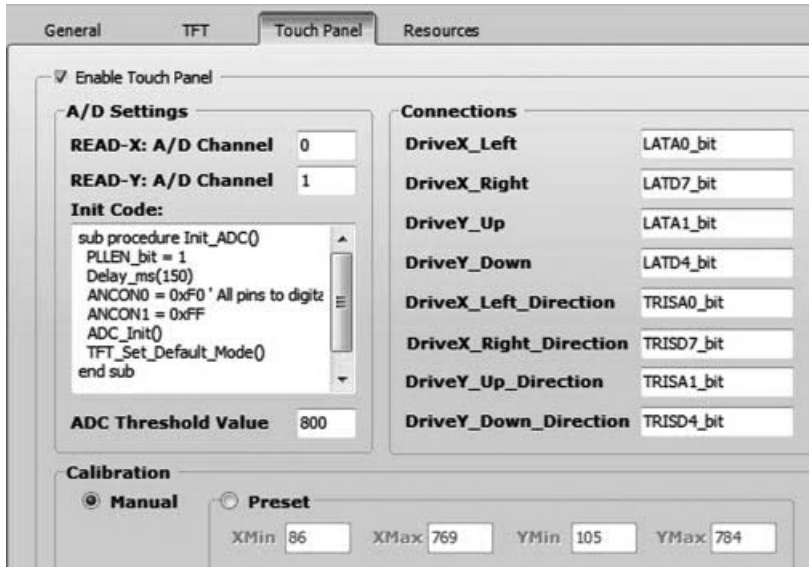


Figure 15.4 Touch Panel settings

and CLR buttons using the 'Rounded Button' tool. Change the names of START and CLR buttons to 'strt' and 'clearkey', respectively. Notice that you can change the colours of the components or the screen background colour as you wish.

Notice that the demo version of the Visual TFT is limited to a maximum of 7 objects on the screen.

Figure 15.5 Adding Components onto the Screen

www.it-ebooks.info

5. *Assign Actions to Components*
We can now add actions to our components. Click on number 1, then double click on 'OnClick' in the Properties window. An empty event function corresponding to this key will be displayed. Enter the following code inside the function:

```
Update(1);
```

Repeat for all the numbers 0 to 9, by changing the number inside the bracket accordingly. Function 'Update' is used to find the final value of the number entered by the user. This function is created as follows:

```
void Update(char d)
{
    char Txt[4];
    sum = 10*sum + d;
    ByteToStr(d, Txt);
    Ltrim(Txt);
    strcat(disp.caption, Txt);
    DrawRoundButton(&disp);
}
```

The final value of the number is stored in variable 'sum'. The digits entered by the user are converted into string and are displayed in the display area of the screen.

Click on the CLR button, then double click on 'OnClick' in the Properties window and enter the following statements inside the function. This function clears the display area of the screen and also clears variable 'sum' to zero:

```
void clearkeyClick()
{
    strcpy(disp.caption, "");
    DrawRoundButton(&disp);
    sum = 0;
}
```

Finally, click on the START button, then double click on 'OnClick' in the Properties window and enter the following statements inside the function. This function is executed inside a loop, where variable 'sum' is decremented by 1 and a 1 second delay is introduced at each iteration. The value of 'sum' is displayed on the display area of the screen as it counts down to zero.

```
void strtClick()
{
    while(sum != 0)
    {
        Delay_Ms(1000);
```

www.it-ebooks.info

```
#include "stopwatch_objects.h"
#include "stopwatch_resources.h"

//---------------------- User code ----------------------//
char sum = 0;
//----------------- End of User code ------------------//

// Event Handlers
void Update(char d)
{
  char Txt[4];
  sum = 10*sum + d;
  ByteToStr(d, Txt);
  Ltrim(Txt);
  strcat(disp.caption, Txt);
  DrawRoundButton(&disp);
}

void no1Click() {
  Update(1);
}

void no2Click() {
  Update(2);
}

void no3Click() {
  Update(3);
}

void no4Click() {
  Update(4);
}

void no5Click() {
  Update(5);
}

void no6Click() {
  Update(6);
}

void no7Click() {
  Update(7);
}

void no8Click() {
  Update(8);
}
```

Figure 15.6 The code associated with the components

www.it-ebooks.info

```
void no9Click() {
  Update(9);
}

void no0Click() {
  Update(0);
}

void clearkeyClick() {
  strcpy(disp.caption, "");
  DrawRoundButton(&disp);
  sum = 0;
}

void strtClick() {
  while(sum != 0)
  {
   Delay_Ms(1000);
   sum--;
   IntToStr(sum, disp.caption);
   DrawRoundButton(&disp);
  }
}
```

Figure 15.6 (*Continued*)

```
        sum-;
        IntToStr(sum, disp.caption);
        DrawRoundButton(&disp);
    }
}
```

Figure 15.6 shows the final value of the user code. Notice that variable 'sum' is declared as a global variable at the beginning of the code.

6. *Generate the Code*
   We are now ready to generate the code for our project. Just click the 'Generate Code' icon in the top menu. You should get a message to say that the code has been generated successfully. The generated code can be seen by clicking the 'Generated Code' at the bottom part of the screen.
7. *Compile the Code and Load to the Microcontroller*

Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before, by clicking the 'Build' icon in the top menu. Make sure that the string library and the conversions libraries are included in the project. You can check this by clicking View -> Library Manager and making sure that the boxes next to 'C_String' and 'Conversions' are checked.

We are now ready to program the microcontroller using the Bootloader software:

- Connect the MikroMMB board to the PC via the USB port.
- Start the MikroMMB Bootloader software on the PC (see Figure 15.7).

www.it-ebooks.info

Figure 15.7 MikroMMB Bootloader software

- Press the RESET button on the MikroMMB board. Then click the Connect' button within 5 seconds. You should see the message 'Connected . . . '. The microcontroller on the MikroMMB board can now be programed.
- Click 'Browse for HEX' and select the '.hex' file of your project.
- Click 'Begin Uploading' to program the microcontroller. You should now see a progress bar as the programming is in progress. When the uploading is finished, click 'OK' to exit.
- Reset the MikroMMB board. The display will start after 5 seconds (if during the first 5 seconds after a RESET there is no communication with a PC, then the microcontroller assumes that this is a normal run and not a programming run).

When the program is run, the user is asked initially to calibrate the screen by touching the bottom-left and upper-right points of the screen. Figure 15.8 shows a typical display from the project where the countdown starts from 25 seconds.

15.2 PROJECT 15.2 – Electronic Book

15.2.1 Project Description

This project describes the design of an electronic book, where a number of pages are displayed on the screen. In this project, there are three pages for simplicity. Two arrow shaped buttons are placed at the bottom of the display. Clicking the right-hand arrow displays the next page (unless it is the last page), while clicking the left-hand arrow displays the previous page (unless it is the first page).

www.it-ebooks.info

Figure 15.8 Typical display from the project

The project is given the name BOOK. The MikroMMB board is used in the design, as in the previous project. The project configuration steps are as in the previous project, and only the changes are given here.

3. *Add Screen and Give it a Name*
   In this project, the four book pages are converted into images and are displayed on the screen. A separate screen is assigned to each page. The screens are named as SCREEN1, SCREEN2 and SCREEN3.
4. *Place components*
   We will now load the pages (images) in our book, with each page loaded to a different screen. Initially, you can use a word processing package (e.g. Word or Notepad) to create the pages. In this example, the Microsoft Publisher was used, with Times New Roman and font size of 8. The number of characters on each line should not exceed 40 and the maximum number of lines should be no more than 8, leaving space for the buttons at the bottom. After creating each page, you should then convert the pages into image format and store as separate files, as these files will be loaded to the screens. If using the Microsoft Publisher, you can save the text as a picture by right clicking on the text.
   In this example, the book consists of three pages, with the following text in each page:

15.2.1.1 PAGE 1

Laboratories are a very important part of every engineering course. Students learn the theory in classes and apply their knowledge into practise by using real equipment in laboratory sessions. For example, electronic engineering students learn the complex theory of transistor amplifiers in the classroom.

www.it-ebooks.info

15.2.1.2 PAGE 2

Simulation is an alternative to real experiments. In control engineering, Matlab is the most widely used software simulation tool. Students can create a model of the system to be simulated by using transfer function blocks, summing points, test inputs, and source and sink devices. The system model can then be simulated by applying inputs and observing the response graphically.

15.2.1.3 PAGE 3

Although laboratory experiments are very useful, they have some problems associated with them:

1. Laboratory equipment can easily be damaged, for example by dropping or by misusing them.
2. The characteristics of real equipment can change with ageing and temperature.
3. Laboratory equipment is costly to purchase and maintain.

In the screen Properties window on the left, set screen colour to White, and orientation to Landscape. Click the Image tool and place it on the screen. Click next to Picture Name in Properties, and select the first page image of the book from the appropriate directory. Position the image to fit the screen. Next, click the Rounded Button tool and place a button at the bottom right-hand side of the image. Change the caption of this button to be 'NEXT ->', as shown in Figure 15.9.

Now, create a new screen by clicking on the green '+' Add Screen tool placed at the top of the menu. Set the screen colour to White and the orientation to Landscape as before. Add the second page image to the screen, and this time place two buttons at the bottom of the screen, as shown in Figure 15.10, so that we can navigate to the next or the previous screens.

Finally, place the last page image and add a button, as shown in Figure 15.11, so that we can navigate back to the previous page.

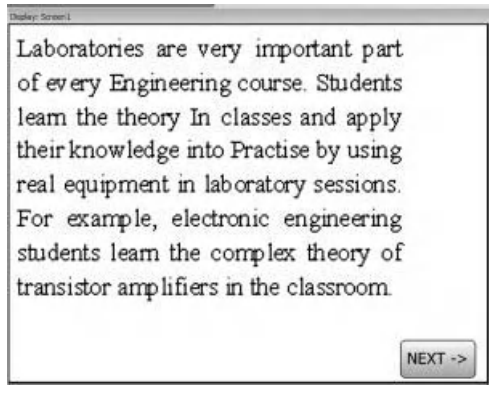

Laboratories are very important part of every Engineering course. Students learn the theory In classes and apply their knowledge into Practise by using real equipment in laboratory sessions. For example, electronic engineering students learn the complex theory of transistor amplifiers in the classroom.
NEXT ->
Display: Screen1

Figure 15.9 Adding the first page image to the screen

www.it-ebooks.info

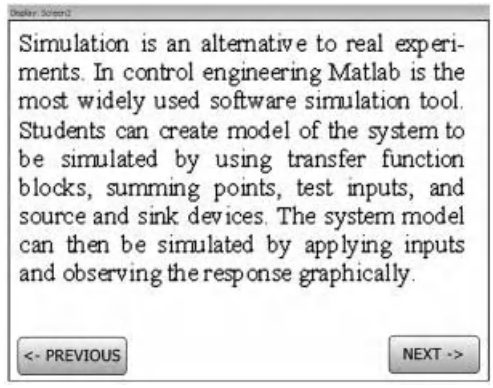

Simulation is an alternative to real experiments. In control engineering Matlab is the most widely used software simulation tool. Students can create model of the system to be simulated by using transfer function blocks, summing points, test inputs, and source and sink devices. The system model can then be simulated by applying inputs and observing the response graphically.
<- PREVIOUS
NEXT ->

Figure 15.10 Adding the second image to the screen

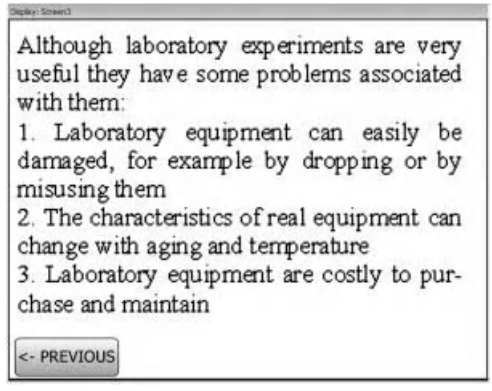

Although laboratory experiments are very useful they have some problems associated with them:
1. Laboratory equipment can easily be damaged, for example by dropping or by misusing them
2. The characteristics of real equipment can change with aging and temperature
3. Laboratory equipment are costly to purchase and maintain
<- PREVIOUS

Figure 15.11 Adding the last image

5. *Assign Actions to Components*
   We can now assign actions to our components. When the display is reset, after calibrating the screen, the first image will automatically be displayed, since the StartScreen property of this screen is set to True. Now, we have to assign an action to the 'NEXT ->' button, so that when this button is pressed, the next image is displayed. Click on the 'NEXT ->' button, and then double click on 'OnClick' in the Properties window. An empty event function corresponding to this button will be displayed. Enter the following code inside the function, so that the second image will be displayed when the button is clicked:

```
DrawScreen(&Screen2);
```

Select Screen2 and click on '<- Previous' button. Double click on 'OnClick' in the Properties window and enter the following code:

```
DrawScreen(&Screen1);
```

www.it-ebooks.info

Similarly, click on the 'NEXT ->' button, double click on 'OnClick' in the Properties window and enter the following code, so that the third image is displayed when the button is pressed:

```
DrawScreen(&Screen3);
```

Finally, select Screen3 and click on '<- Previous' button. Double click on 'OnClick' in the Properties window and enter the following code, so that the previous page image (image 2) is displayed when the button is clicked:

```
DrawScreen(&Screen2);
```

Figure 15.12 shows all the user action codes.

6. *Generate the Code*
   We are now ready to generate the code for our project. Just click the 'Generate Code' icon in the top menu. You should get a message to say that the code has been generated successfully. The generated code can be seen by clicking the 'Generated Code' at the bottom part of the screen.
7. *Compile the Code and Load to the Microcontroller*

Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before by clicking the 'Build' icon in the top menu. Load the

```
#include "BOOK_objects.h"
#include "BOOK_resources.h"

//--------------------- User code ---------------------//

//---------------- End of User code ------------------//

// Event Handlers

void ButtonRound1Click() {
  DrawScreen(&Screen2);
}

void ButtonRound3Click() {
  DrawScreen(&Screen1);
}

void ButtonRound2Click() {
  DrawScreen(&Screen3);
}


void ButtonRound4Click() {
  DrawScreen(&Screen2);
}
```

Figure 15.12 User action codes

www.it-ebooks.info

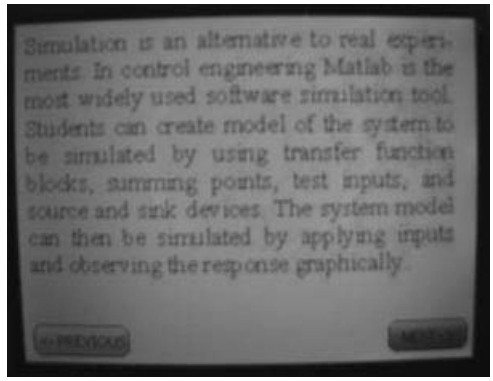

Simulation is an alternative to real experiments. In control engineering Matlab is the most widely used software simulation tool. Students can create model of the system to be simulated by using transfer function blocks, summing points, test inputs, and source and sink devices. The system model can then be simulated by applying inputs and observing the response graphically.

Figure 15.13 A typical display

program into the program memory of the microcontroller, using the Bootloader software as described in the previous project. Press the reset button on the display board to start the project. After calibrating the screen, you should see the first image displayed and you should be able to navigate to other images by clicking the appropriate buttons.
Figure 15.13 shows a typical display on the screen.

15.3 PROJECT 15.3 – Picture Show

15.3.1 Project Description

This project describes the design of a slide show where a number of images (four in this project) are stored on a mikroSD card and are displayed on the TFT screen with a delay between each display. The MikroMMB board is used in the design, as in the previous project. The project is named 'PICTURE_SHOW'. The project configuration steps are as in the previous project. In addition, the Resources window in Settings should be configured as below (see Figure 15.14), to define the interface between the mikroSD card and the MikroMMB board (you may find that this window is already configured when the external resource is clicked):

```
Store resources:    Externally
SelectMedia:        MMC

Global Declarations:

sbit Mmc_Chip_Select at LATD0_bit;
sbit Mmc_Chip_Select_Direction at TRISD0_bit;

Init Code:
SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64, _SPI_DATA_SAMPLE_MIDDLE,
_SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
Delay_ms(10);
```

www.it-ebooks.info

```
Get Data Code:
char* TFT_Get_Data(unsigned long offset, unsigned int count, unsigned int
  *num) {
unsigned long start_sector;
unsigned int pos;

start_sector = Mmc_Get_File_Write_Sector() + offset/512;
pos = (unsigned long)offset%512;

if(start_sector == currentSector + 1) {
Mmc_Multi_Read_Sector(Ext_Data_Buffer);
currentSector = start_sector;
} else if (start_sector != currentSector) {
if(currentSector != -1)
Mmc_Multi_Read_Stop();
Mmc_Multi_Read_Start(start_sector);
Mmc_Multi_Read_Sector(Ext_Data_Buffer);
currentSector = start_sector;
}

if(count>512-pos)
  *num = 512-pos;
else
  *num = count;
return Ext_Data_Buffer + pos;
}
```

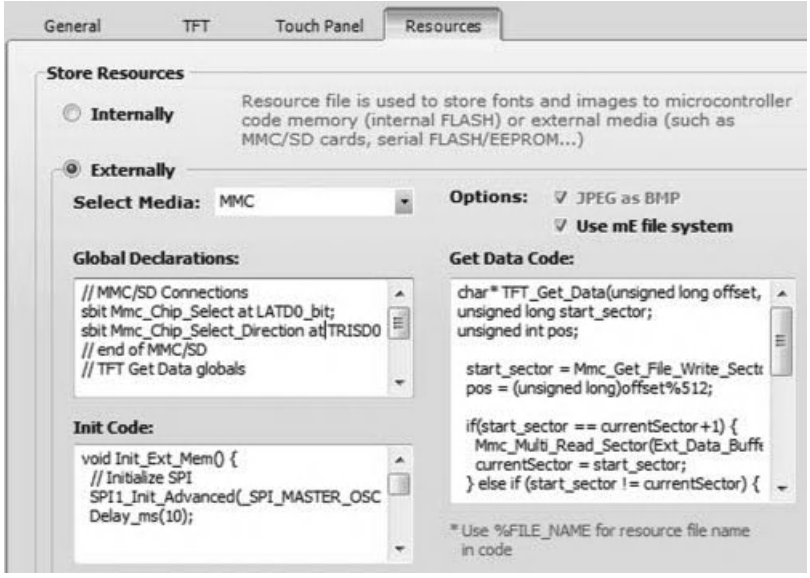


Figure 15.14 Resources window

www.it-ebooks.info

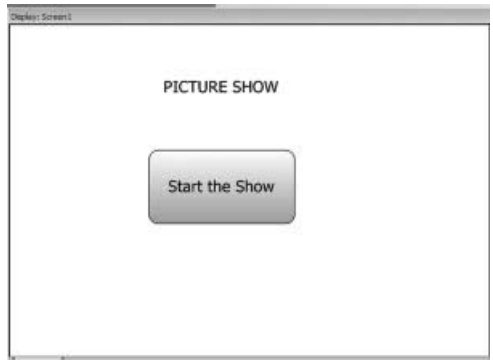


Figure 15.15 Screen1 layout

3. *Add Screen and Give it a Name*
   In this project, we have two screens: the startup screen, and the screen where the pictures are to be displayed. The startup screen will simply have a button and pressing this button will start the picture show. Rename the startup screen as Screen1, and set screen colour to White, with orientation set to Landscape.
4. *Place components*
   Place a Label on Screen1 and change its caption to 'PICTURE SHOW'. Also, place a Round button on this screen, and change its caption to 'Start the Show'. Figure 15.15 shows the Screen1 layout.

   Create a new screen by clicking the green '+' toolbar. Rename this as Screen2 and set its orientation to Landscape. We will now load the images in our picture show. Make sure that the images are Bitmap images with extensions '.BMP' and that they have the pixel sizes 320×240. Rotate the pictures by 90° clockwise, so that they will fit the screen nicely. There are many programs available to rotate images and to convert them to different sizes (e.g. FastStone Image Viewer).

   In this example, four images are selected at random from the Internet for demonstration purposes. These images are converted to 320×240 pixels and then rotated by 90° and stored in a directory on the PC.

   Click the Image tool and place it on the screen. Click next to Picture Name and select the first picture of the picture show from the appropriate directory. Position the image to fit the screen (see Figure 15.16). Next, click the Image tool again and place it on the screen on top of the existing image. Select the second image from the appropriate directory by clicking next to Picture Name and again position the image to fit the screen. Repeat this process until all the images are placed on the screen. As you place images on the screen, only the last image will be visible. At the end, click the list box under the Components window on the middle left part of the screen. You should see all the image names that are loaded to the screen.

   Click 'Generate Code' in the top menu of the screen. All the images are now compiled and stored in a file called the Resource file. This file is used to store fonts and images in the microcontroller code memory or in an external memory device, such as a mikroSD card. You should now format a mikroSD card with the FAT16 filing system and copy the Resource file to the card. The SD card should be less than 2 GB in size. Click the button

www.it-ebooks.info

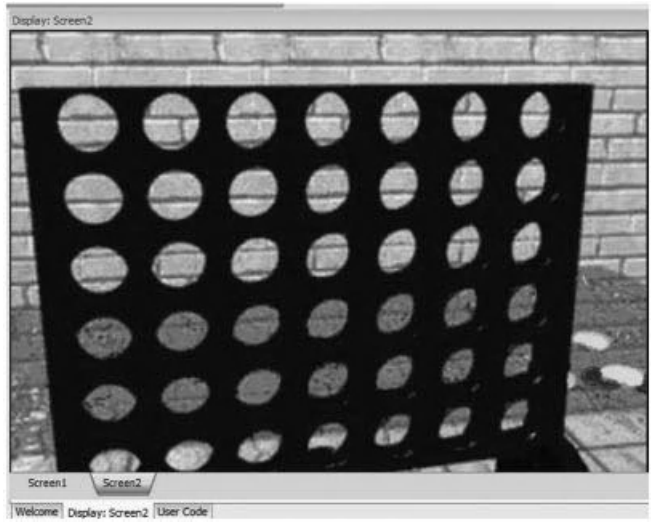


Figure 15.16 Adding an image to the screen

named 'Locate resource file', located at the bottom right-hand side of the screen, to find the resource file. The resource file is the one with extension '.RES' attached to the 8 character project name. For this project, the resource file is 'PICTURE_.RES', and this file must be copied to the mikroSD card.

5. *Assign Actions to Components*
We can now assign actions to our components. When the display is reset, and after calibrating the screen, the pictures will be displayed with a delay between each display.

Before assigning actions to components, we need to know the external addresses of the images in our project. This can be obtained from the file PICTURE_SHOW_RESOURCES.C in our project. Click 'Locate resource file' and then right click on this file and choose the edit option to see the contents of the file. The addresses of the images for this project are found to be:

```
const unsigned long image1_bmp = 0x0000071C;
const unsigned long image2_bmp = 0x00025F22;
const unsigned long image3_bmp = 0x0004B728;
const unsigned long image4_bmp = 0x00070F2E;
```

We shall be using these addresses to display the images.

Click on the Round button in Screen1. Double click 'OnClick' in the Properties window to assign action to this button. Enter the following code in the function body:

```
void ButtonRound1Click() {
    for(;;)
    {
         TFT_Ext_Image(0, 0, Images[Cnt], 1);
          Delay_Ms(10 000);
```

www.it-ebooks.info

```
            Cnt++;
        if(Cnt == 4)Cnt =0;
      }
    }
```

and enter the following code in the user code section:

```
const unsigned long Images[] = {0x0000071C, 0x00025F22, 0x0004B728,
 0x00070F2E};
unsigned char Cnt = 0;
```

The addresses of the images are stored in an array called *Images*, and a variable called *Cnt* is declared and initialised to 0. When the 'Start the Show' button is clicked, function ButtonRound1Click will be called. Inside this function, an endless loop is formed using a *for* statement. TFT library function TFT_Ext_Image is called to display an image. The address of the image to be displayed is specified by indexing array *Images* with variable *Cnt*. Thus, initially the first image is displayed. Then after 10 seconds delay, variable *Cnt* is incremented by 1 and the second image is displayed, and so on. After displaying the last image (Cnt = 4), variable *Cnt* is reset to 0 to display the first image again and this process continues forever.

Figure 15.17 shows the final form of the user action code.

6. *Compile the Code and Load to the Microcontroller*

Click the icon 'Start Compiler' in the top menu to start the mikroC Pro for PIC compiler. Compile the program as before, by clicking the 'Build' icon in the top menu. Make sure that the SD card library is included in the build process. To check this, display the Library Manager by clicking View -> Library Manager at the top menu. Tick boxes 'Mmc', 'Mmc_FAT16' and 'Mmc_FAT16_Config', if they are not already ticked. Now, we are ready

```
#include "PICTURE_SHOW_objects.h"
#include "PICTURE_SHOW_resources.h"

//--------------------- User code ---------------------//
const unsigned long images[] = {0x0000071C,0x00025F22,0x0004B728,0x00070F2E};
unsigned char Cnt = 0;
//----------------- End of User code ------------------//

// Event Handlers

void ButtonRound1Click() {
 for(;;)
 {
    TFT_Ext_Image(0, 0, Images[Cnt], 1);
    Delay_Ms(10000);
    Cnt++;
    if(Cnt == 4)Cnt =0;
 }
}
```

Figure 15.17 The user action code

www.it-ebooks.info

to program the microcontroller using the Bootloader software, as described in the previous project.

After the programming is finished, reset the MikroMMB board. The display will start after 5 seconds. When the program is run, the user is asked initially to calibrate the screen by touching the bottom-left and upper-right points of the screen. Now, the pictures will be displayed as a picture show, with 10 seconds delay between each output.

15.4 Summary

This chapter has described how to use the Visual TFT software. Visual TFT is a standalone software package used for the development of TFT based colour GLCD projects. The software can be used with a number of TFT based GLCD modules. In this chapter, the 'MikroMMB board for PIC18FJ' has been used as the hardware development environment. Several tested and working projects have been given in the chapter, to show how to design colour graphics based projects.

Exercises

15.1 Design an integer calculator using the Visual TFT software and the MikroMMB board for PIC18FJ. Experiment using different colours in your screen design.

15.2 Describe the steps necessary for using the Visual TFT software to design a GLCD based project.

15.3 Design a slide show using the Visual TFT software and the MikroMMB board for PIC18FJ. Use your family photos in the slide show.

15.4 Design a calculator to convert °C into °F and vice versa, using the Visual TFT software and the MikroMMB board for PIC18FJ. A button should be available on the screen to choose the type of conversion required.

www.it-ebooks.info

Bibliography

1. Microchip Inc. web site: http://www.microchip.com.
2. Mikroelektronika web site: http://www.mikroe.com.
3. Custom Computer Systems Inc* web site: http://www.ccsinfo.com.
4. mikroEngineering Labs Inc web site: http://www.melabs.com.
5. Futurlec web site: http://www.futurlec.com.
6. SensirionInc web site: http://www.sensirion.com.
7. LM35DZ data sheet web site: http://www.national.com.
8. Hi-Tech Software web site: http://www.htsoft.com.
9. KS0108 controller data sheet web site: http://www.techtoys.com.hk/Displays/JHD12864J/ks0108.pdf

*(*readers who link to web site* www.ccsinfo.com*/DIbookwill be given to a discount by Custom Computer Systems Inc for their hardware and software products*)

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.

© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

Index

ADC, 5, 43
ADCON0, 45
ADCON1, 45
ADCON2, 47
ADRESH, 47
ADRESL, 47
Alphanumeric LED, 159
ALU, 5
Analog comparator, 5
Analog to digital converter, 5, 43
Anode, 152
ANSI C library, 109
Array pointer, 75
Arrays, 70
  numeric, 70
  character, 72
  of strings, 73
Arrays of strings, 73
Arithmetic operator, 81
Aspect ratio, 15
Assignment operator, 86

BASIC, 59
BEGIN, 206
Bi-colour LED, 154
Bitmap editor, 198
Bitmap image, 347
Bits of a variable, 69
bitwise operator, 83
bit type, 70
break, 92
Break point, 124
Brightness, 15
Brown-out detector, 5
Bus, 5
Bootloader, 461
Built-in functions, 108
Busy flag, 174
Button, 238
ByteToStr, 111

CAN, 6
Cathode, 152
CCP1CON, 54
CCS C compiler, 60
char, 65
Character arrays, 72
CGRAM, 168
CGROM, 168
CISC, 6
Clock, 6
Clock_Khz, 108
Clock_Mhz, 108
Code assistant, 112
Code editor, 42
Code explorer window, 115
Comments, 61
Common anode, 157
Common cathode, 157
Conditional operator, 87
CONFIG1H, 48
CONFIG2H, 49
Configuration register, 48
const, 66

Using LEDs, LCDs and GLCDs in Microcontroller Projects, First Edition. Dogan Ibrahim.

© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

www.it-ebooks.info

Constant string, 73
Constants, 66
Contrast ratio, 15
CPU, 6
Crystal, 24
Current limiting resistor, 152
Current sinking, 220
Current sourcing, 220
Custom font, 315

Data memory, 21
DDRAM, 169
Debounce, 238
Debugger, 145
Delay_ms, 108
Delay_us, 108
Digital voltmeter, 325
Display software tools, 143
  font creation tools, 143
  display library tools, 144
  visual display tools, 144
Display timing, 171
do, 97
Dot pitch, 15
DSTN, 14
Duty cycle, 54

EasyPIC6, 137
EasyPIC7, 139
EEPROM, 6
else, 91
END, 206
ENDDO, 207
enum, 68
Enumarated constant, 67
EPROM, 6
Escape sequence, 68
Ethernet, 7
Event counter, 285, 292
External clock, 28
External reset, 31

Flash memory, 7
float, 65
Floating point, 65
Floating point constant, 67
Flow chart, 206
Flow of control, 90
for, 99

FOREVER, 208
Forward current, 151
FSTN, 14
Functions, 101
  built-in, 108
  name, 102
  passing arrays to, 106
  passing parameters to, 104
  type, 102
  void, 103
Function prototype, 102

GLCD, 185
  Block diagram, 186
  Connection to microcontroller, 188
  Display coordinates, 189
  Operation, 187
  Pin configuration, 186
  Structure, 188
Glcd_Box, 193
Glcd_Circle, 193
Glcd_Circle_Fill, 194
Glcd_Dot, 191
Glcd_Fill, 190
Glcd_Init, 189
Glcd_Line, 191
Glcd_Rectangle, 192
Glcd_Rectangle_Round_Edges, 192
Glcd_Rectangle_Round_Edges_Fill, 192
Glcd_Set_Font, 194
Glcd_Set_Font_Adv, 194
Glcd_Set_Side, 190
Glcd_Set_X, 190
Glcd_H_Line, 191
Glcd_Image, 196
Glcd_V_Line, 191
Glcd_Write_Char, 195
Glcd_Write_Char_Adv, 195
Glcd_Write_Const_Text_Adv, 196
Glcd_Write_Data, 190
Glcd_Write_Text, 195
Glcd_Write_Text_Adv, 195
Goto, 101

Harvard architecture, 7, 17
HD44780 controller, 165
Hi, 108
Higher, 108

www.it-ebooks.info

Highest, 108
Humidity sensor, 385

ICD-U40, 61
Idle mode, 7
if, 91
In-circuit debugger, 145
int, 65
INTCON, 50, 51
INTCON2, 50, 52
INTCON3, 50, 52
Integer constant, 66
Internal oscillator, 27
Interrupts, 7, 49
Interrupt processing, 106
IPEN, 53
I/O port, 31

Keypad, 446
Keypad_init, 337
Keypad_key_click, 337
Keypad_key_press, 337
KS107/108, 347

LATA, 33
LATB, 35
LATC, 37
LCD, 165
  1 × 16, 166
  4 × 16, 166
  character set, 168
  contrast, 167
  pins, 166
Lcd_Init, 180
Lcd_Chr, 181
Lcd_Chr_Cp, 181
Lcd_Cmd, 182
Lcd_Out, 181
LED, 151
LED
  14 segment, 160
  alphanumeric, 159
  common anode, 157
  common cathode, 157
  construction, 152
  decoding, 162
  editor, 163
  encoding, 158
  microC PRO, 163
  multi digit, 159
LED bar, 155
LED candle, 264
LED colours, 153
LED dice, 240
LED dot matrix, 156
LED sizes, 154
Library manager window, 115
LM35DZ, 330
Lo, 108
Logical operators, 82
long, 65

Math library, 110
MCLR, 30
Message window, 115
mikroICD, 145
mikroMedia, 142, 454
  component side, 143
MikroMMB board for PIC18FJ, 454

Numeric arrays, 70

OLED, 12
Operators, 80
  arithmetic, 81
  assignment, 86
  bitwise, 83
  conditional, 87
  logical, 82
  relational, 85
OSC1, 24
OSC2, 24
OSCCON, 28
Oscillator configuration, 24

Paint program, 347
Parallel I/O port, 31
Parameter assistant, 114
PASCAL, 59
PDL, 205
Phase locked loop, 28
PIC18 explorer board, 132
PIC18F4XK20 starter kit, 134
PICC18 compiler, 60
PICDEM 4, 135
Pipeline, 8
PLL, 28
Pointer arithmetic, 74

www.it-ebooks.info

Pointers, 73
  array, 75
  in string operations, 76
POR, 30
PORT A, 33
PORT B, 35
PORT C, 36
PORT E, 37
Power on reset, 8
Pre-processor, 87
  #define, 88
  #elif, 90
  #endif, 90
  #if, 90
  #ifndef, 88
  #include, 89
  #undef, 88
Pressure sensor, 330
Program memory, 21
Project manager window, 117
Project settings window, 115
PROM, 8
Pulse width modulator, 53
PWM, 53
PWM duty cycle, 54
PWM period, 53

RAM, 8
RCON, 50
Real time clock, 8
Reflective, 15
Register, 9
Relational operators, 85
REPEAT, 209
Repetition, 95
Reset, 30
Reserved names, 64
Resistive touch screen, 200
Resolution, 15
Resonator, 24
Response time, 15
RISC, 9
ROM, 9
Rotating LEDs, 229
Routine list window, 115

sbit, 70
Selection statements, 91
Seven segment LDE, 156
SHT11, 385
signed char, 65
signed int, 65
Simulator, 123
sizeof, 79
Sleep mode, 9
SmartGLCD, 430
  back panel, 142
  front panel, 142
STN, 14
Strings, 72
String constants, 67
Structures, 76
  accessing, 78
  arrays of, 79
  bit fields, 79
  copying, 78
  initialising, 78
  size of, 79
Super bundle development kit, 133
Supply voltage, 10
switch, 92

Temperature sensor, 330
TFT, 14
TIMER0, 38
TIMER1, 40
TIMER2, 41
TIMER3, 43
Timer module, 38
TMR0L, 38
TMR0H, 38
Touch screen, 199, 401
Transflective, 15
Transmissive, 15
Tri-colour LED, 155
TRISA, 33
TRISB, 35
TRISC, 37
Twisted nematic, 14

Unions, 80
unsigned char, 65
unsigned int, 65
unsigned long int, 65
UNTIL, 209
USART terminal, 127
USB, 10

Variable name, 63
Variable type, 64

www.it-ebooks.info

Vdelay_ms, 108
View angle, 15
Visual GLCD, 419
Visual TFT, 453
Void function, 103
Volatile variables, 69

Watchdog, 10
White space, 63
while, 95
WREG, 19

XT, 24

www.it-ebooks.info